

Babel

Code

Version 24.10
2024/09/18

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Localization and
internationalization

Unicode

TEX

pdfTEX

LuaTEX

XeTEX

Contents

1	Identification and loading of required files	3
2	locale directory	3
3	Tools	3
3.1	Multiple languages	7
3.2	The Package File (\LaTeX , <code>babel.sty</code>)	8
3.3	<code>base</code>	9
3.4	<code>key=value</code> options and other general option	10
3.5	Conditional loading of shorthands	11
3.6	Interlude for Plain	13
4	Multiple languages	13
4.1	Selecting the language	15
4.2	Errors	23
4.3	Hooks	25
4.4	Setting up language files	27
4.5	Shorthands	29
4.6	Language attributes	38
4.7	Support for saving macro definitions	39
4.8	Short tags	41
4.9	Hyphens	41
4.10	Multiencoding strings	43
4.11	Macros common to a number of languages	48
4.12	Making glyphs available	49
4.12.1	Quotation marks	49
4.12.2	Letters	50
4.12.3	Shorthands for quotation marks	51
4.12.4	Umlauts and tremas	52
4.13	Layout	53
4.14	Load engine specific macros	53
4.15	Creating and modifying languages	54
5	Adjusting the Babel behavior	77
5.1	Cross referencing macros	79
5.2	Marks	82
5.3	Preventing clashes with other packages	82
5.3.1	<code>ifthen</code>	82
5.3.2	<code>varioref</code>	83
5.3.3	<code>hhline</code>	84
5.4	Encoding and fonts	84
5.5	Basic bidi support	86
5.6	Local Language Configuration	89
5.7	Language options	89
6	The kernel of Babel (<code>babel.def</code>, <code>common</code>)	93
7	Loading hyphenation patterns	96
8	Font handling with <code>fontspec</code>	100
9	Hooks for XeTeX and LuaTeX	103
9.1	XeTeX	103

10	Support for interchar	105
10.1	Layout	107
10.2	8-bit TeX	109
10.3	LuaTeX	110
10.4	Southeast Asian scripts	116
10.5	CJK line breaking	117
10.6	Arabic justification	119
10.7	Common stuff	124
10.8	Automatic fonts and ids switching	124
10.9	Bidi	130
10.10	Layout	132
10.11	Lua: transforms	140
10.12	Lua: Auto bidi with basic and basic-r	149
11	Data for CJK	161
12	The ‘nil’ language	161
13	Calendars	162
13.1	Islamic	162
13.2	Hebrew	164
13.3	Persian	168
13.4	Coptic and Ethiopic	168
13.5	Buddhist	169
14	Support for Plain TeX (plain.def)	170
14.1	Not renaming hyphen.tex	170
14.2	Emulating some L ^A T _E X features	171
14.3	General tools	171
14.4	Encoding related macros	175
15	Acknowledgements	178

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 L^AT_EX package, which set options and load language styles.

babel.def is loaded by Plain.

switch.def defines macros to set and switch languages (it loads part babel.def).

plain.def is not used, and just loads babel.def, for compatibility.

hyphen.cfg is the file to be used when generating the formats to load hyphenation patterns.

There some additional tex, def and lua files.

The babel installer extends docstrip with a few “pseudo-guards” to set “variables” used at installation time. They are used with <@name> at the appropriate places in the source code and defined with either <<name=value>>, or with a series of lines between <<*name>> and <</name>>. The latter is cumulative (eg, with *More package options*). That brings a little bit of literate programming. The guards <-name> and <+name> have been redefined, too. See babel.ins for further details.

2 locale directory

A required component of babel is a set of ini files with basic definitions for about 300 languages. They are distributed as a separate zip file, not packed as dtx. Many of them are essentially finished (except bugs and mistakes, of course). Some of them are still incomplete (but they will be usable), and there are some omissions (eg, there are no geographic areas in Spanish). Not all include LICR variants.

babel-*.ini files contain the actual data; babel-*.tex files are basically proxies to the corresponding ini files.

See [Keys in ini files](#) in the the babel site.

3 Tools

```
1 <<version=24.10>>
2 <<date=2024/09/18>>
```

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 L^AT_EX 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@cl#1{\csname bbl@#1\@languagenamename\endcsname}
18 \def\bbl@loop#1#2#3{\bbl@loop#1#3#2,\@nnil,}
```

```

19 \def\bbloopx#1#2{\expandafter\bbloop\expandafter#1\expandafter{#2}}
20 \def\bbloop#1#2#3,{%
21   \ifx\@nnil#3\relax\else
22     \def#1{#3}#2\bb@afterfi\bbloop#1{#2}%
23   \fi}
24 \def\bb@for#1#2#3{\bbloopx#1{#2}{\ifx#1\@empty\else#3\fi}}

```

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

```

`\bb@afterelse` Because the code that is used in the handling of active characters may need to look ahead, we take `\bb@afterfi` 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\bb@afterelse#1\else#2\fi{\fi#1}
32 \long\def\bb@afterfi#1\fi{\fi#1}

```

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

```

`\bb@trim` The following piece of code is stolen (with some changes) from `keyval`, by David Carlisle. It defines two macros: `\bb@trim` and `\bb@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\bb@tempa#1{%
44   \long\def\bb@trim##1##2{%
45     \futurelet\bb@trim@a\bb@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
46   \def\bb@trim@c{%
47     \ifx\bb@trim@a\@sptoken
48       \expandafter\bb@trim@b
49     \else
50       \expandafter\bb@trim@b\expandafter#1%
51     \fi}%
52   \long\def\bb@trim@b#1##1 \@nil{\bb@trim@i##1}}
53 \bb@tempa{ }
54 \long\def\bb@trim@i#1\@nil#2\relax#3{#3{#1}}
55 \long\def\bb@trim@def#1{\bb@trim{def#1}}

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

101 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
102   \toks@{}}
103 \def\bbl@replace@aux##1#2##2#2{%
104   \ifx\bbl@nil##2%
105   \toks@\expandafter{\the\toks@##1}%
106   \else

```

```

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

```

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

```

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

```

Two further tools. \bbl@ifsamestring first expand its arguments and then compare their expansion (sanitized, so that the catcodes do not matter). \bbl@engine takes the following values: 0 is pdf_{TEX}, 1 is luatex, and 2 is xetex. You may use the latter 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\bb@bsphack{%
163   \ifhmode
164     \hskip\z@skip
165     \def\bb@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
166   \else
167     \let\bb@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\bb@cased{%
170   \ifx\oe\OE
171     \expandafter\in@\expandafter
172       {\expandafter\OE\expandafter}\expandafter{\oe}%
173   \ifin@
174     \bb@afterelse\expandafter\MakeUppercase
175   \else
176     \bb@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\bb@extras@wrap#1#2#3{% 1:in-test, 2:before, 3:after
182   \toks@\expandafter\expandafter\expandafter{%
183     \csname extras\languagename\endcsname}%
184   \bb@exp{\in@{#1}}{\the\toks@}}%
185   \ifin@\else
186     \@temptokena{#2}%
187     \edef\bb@tempc{\the\@temptokena\the\toks@}%
188     \toks@\expandafter{\bb@tempc#3}%
189     \expandafter\edef\csname extras\languagename\endcsname{\the\toks@}%
190   \fi}
191 <</Basic macros>>

```

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

```

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

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

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

```

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

```

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

```

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

```

3.3 base

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

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

```

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

```

3.4 key=value options and other general option

The following macros extract language modifiers, and only real package options are kept in the option list. Modifiers are saved and assigned to `\BabelModifiers` at `\bbl@load@language`; when no modifiers have been given, the former is `\relax`.

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

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

```
325 \DeclareOption{KeepShorthandsActive}{}
326 \DeclareOption{activeacute}{}
327 \DeclareOption{activegrave}{}
328 \DeclareOption{debug}{}
329 \DeclareOption{noconfigs}{}
330 \DeclareOption{showlanguages}{}
331 \DeclareOption{silent}{}
332 \DeclareOption{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@} % second = 2
336 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % second + 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

$\langle key \rangle = \langle value \rangle$, the second one loads the requested languages, except the main one if set with the key main, and the third one loads the latter. First, we “flag” valid keys with a nil value.

```

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{bad-package-option}{#1}{#2}{}%
355   \fi}

```

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

```

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

```

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

```

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

```

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=...`

```

377 \bbl@trace{Conditional loading of shorthands}
378 \def\bbl@sh@string#1{%
379   \ifx#1\@empty\else
380     \ifx#1t\string~%
381     \else\ifx#1c\string,%
382     \else\string#1%
383   \fi\fi
384   \expandafter\bbl@sh@string
385 \fi}
386 \ifx\bbl@opt@shorthands\@nnil
387   \def\bbl@ifshorthand#1#2#3{#2}%
388 \else\ifx\bbl@opt@shorthands\@empty

```

```

389 \def\bb@ifshorthand#1#2#3{#3}%
390 \else

```

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

```

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

```

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

```

398 \edef\bb@opt@shorthands{%
399 \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.

```

400 \bb@ifshorthand{'}%
401 {\PassOptionsToPackage{activeacute}{babel}}{}
402 \bb@ifshorthand{`}%
403 {\PassOptionsToPackage{activegrave}{babel}}{}
404 \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.

```

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

```

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

```

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

```

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

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

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

```
443 </core>
444 <*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.

```
445 \def\bb@l@version{<@version@>}
446 \def\bb@l@date{<@date@>}
447 <@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.

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

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

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

```

473     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
474     \lowercase\expandafter{\bbl@tempd}}}%
475     \@empty
476     \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
477     \bbl@tempd
478     \bbl@exp{\bbl@usehooks{language}{\language}{#1}}
479 \def\bbl@iflanguage#1{%
480   \ifundefined{l@#1}{\@noLanerr{#1}\@gobble}\@firstofone}

```

After a name has been ‘fixed’, the selectors will try to load the language. If even the fixed name is not defined, will load it on the fly, either based on its name, or if activated, its 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`.

```

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

```

```

530     {\edef\language{\nameuse{bbl@bcp@map@\language}}}%
531 \ifbbl@bcp@allowed
532   \expandafter\ifx\csname date\language\endcsname\relax
533     \expandafter
534     \bbl@bcp@lookup\language-\@empty-\@empty-\@empty@@
535     \ifx\bbl@bcp\relax\else % Returned by \bbl@bcp@lookup
536       \edef\language{\bbl@bcp@prefix\bbl@bcp}%
537       \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
538       \expandafter\ifx\csname date\language\endcsname\relax
539         \let\bbl@initoload\bbl@bcp
540         \bbl@exp{\babelprovide[\bbl@autoload@bcptoptions]{\language}}%
541         \let\bbl@initoload\relax
542       \fi
543       \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
544     \fi
545   \fi
546 \fi
547 \expandafter\ifx\csname date\language\endcsname\relax
548   \IfFileExists{babel-\language.tex}%
549   {\bbl@exp{\babelprovide[\bbl@autoload@options]{\language}}}%
550   {}%
551 \fi}
552 </package>
553 <*package | 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.

```

554 \def\iflanguage#1{%
555   \bbl@iflanguage{#1}{%
556     \ifnum\csname l@#1\endcsname=\language
557       \expandafter\@firstoftwo
558     \else
559       \expandafter\@secondoftwo
560     \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.

```

561 \let\bbl@select@type\z@
562 \edef\selectlanguage{%
563   \noexpand\protect
564   \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`.

```

565 \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.

```

566 \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.

```
567 \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`

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

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

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

```
580 \def\bbl@pop@lang#1+#2\@@{%
581   \edef\languagename{#1}%
582   \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).

```
583 \let\bbl@ifrestoring\@secondoftwo
584 \def\bbl@pop@language{%
585   \expandafter\bbl@pop@lang\bbl@language@stack\@@
586   \let\bbl@ifrestoring\@firstoftwo
587   \expandafter\bbl@set@language\expandafter{\languagename}%
588   \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).

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

```

603     }%
604     \fi}%
605     }%
606     \chardef\localeid\babel@cl{id@}}

```

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

```

607 \expandafter\def\csname selectlanguage \endcsname#1{%
608   \ifnum\babel@hymapsel=\@ccclv\let\babel@hymapsel\tw@\fi
609   \babel@push@language
610   \aftergroup\babel@pop@language
611   \babel@set@language{#1}}
612 \let\endselectlanguage\relax

```

`\babel@set@language` The macro `\babel@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. `\babel@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).

```

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

```

```

650 \fi
651 %^A TODO. name@map must be here?
652 \bbl@provide@locale
653 \bbl@iflanguage\languagename{%
654   \let\bbl@select@type\z@
655   \expandafter\bbl@switch\expandafter{\languagename}}
656 \def\babel@aux#1#2{%
657   \select@language{#1}%
658   \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
659     \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}%^A TODO - plain?
660 \def\babel@toc#1#2{%
661   \select@language{#1}}

```

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

The name of the language is stored in the control sequence `\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` (*language*) command at definition time by expanding the `\csname` primitive. Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of `\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 `\<language>hyphenmins` is defined. If it is not, we set default values (2 and 3), otherwise the values in `\<language>hyphenmins` will be used.

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

```

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

```

```

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

```

otherlanguage (env.) The otherlanguage environment can be used as an alternative to using the \selectlanguage declarative command. The \ignorespaces command is necessary to hide the environment when it is

entered in horizontal mode.

```
758 \long\def\otherlanguage#1{%
759   \def\bbl@selectorname{other}%
760   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
761   \csname selectlanguage \endcsname{#1}%
762   \ignorespaces}
```

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

```
763 \long\def\endotherlanguage{\@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`.

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

```
771 \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{language}` command doesn’t make any `\global` changes. The coding is very similar to part of `\selectlanguage`.

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

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

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

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

```
772 \providecommand\bbl@beforeforeign{}
773 \edef\foreignlanguage{%
774   \noexpand\protect
775   \expandafter\noexpand\csname foreignlanguage \endcsname}
776 \expandafter\def\csname foreignlanguage \endcsname{%
777   \@ifstar\bbl@foreign@s\bbl@foreign@x}
778 \providecommand\bbl@foreign@x[3][]{%
779   \begingroup
780   \def\bbl@selectorname{foreign}%
781   \def\bbl@select@opts{#1}%
782   \let\BabelText\@firstofone
783   \bbl@beforeforeign
784   \foreign@language{#2}%
785   \bbl@usehooks{foreign}{}%
786   \BabelText{#3}% Now in horizontal mode!
787   \endgroup}
```

```

788 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@par
789 \begingroup
790   {\par}%
791   \def\bbl@selectorname{foreign*}%
792   \let\bbl@select@opts\@empty
793   \let\bbl@BabelText\@firstofone
794   \foreign@language{#1}%
795   \bbl@usehooks{foreign*}{}%
796   \bbl@dirparastext
797   \BabelText{#2}% Still in vertical mode!
798   {\par}%
799 \endgroup}
800 \providecommand\bbl@wrapText[1]{%
801   \def\bbl@tempa{\def\bbl@BabelText###1}%
802   \expandafter\bbl@tempa\expandafter{\BabelText{#1}}}

```

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

```

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

```

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

```

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

```

824 \let\bbl@hyphlist\@empty
825 \let\bbl@hyphenation@relax
826 \let\bbl@pttnlist\@empty
827 \let\bbl@patterns@relax
828 \let\bbl@hymapsel=\@cclv
829 \def\bbl@patterns#1{%
830   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
831     \csname l@#1\endcsname
832     \edef\bbl@tempa{#1}%
833   \else
834     \csname l@#1:\f@encoding\endcsname
835     \edef\bbl@tempa{#1:\f@encoding}%

```

```

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

```

852 \def\hyphenrules#1{%
853 \edef\bb@tempf{#1}%
854 \bb@fixname\bb@tempf
855 \bb@iflanguage\bb@tempf{%
856 \expandafter\bb@patterns\expandafter{\bb@tempf}%
857 \ifx\languageshorthands\@undefined\else
858 \languageshorthands{none}%
859 \fi
860 \expandafter\ifx\csname\bb@tempf hyphenmins\endcsname\relax
861 \set@hyphenmins\tw@\thr@\relax
862 \else
863 \expandafter\expandafter\expandafter\set@hyphenmins
864 \csname\bb@tempf hyphenmins\endcsname\relax
865 \fi}}
866 \let\endhyphenrules\@empty

```

`\providehyphenmins` The macro `\providehyphenmins` should be used in the language definition files to provide a *default* setting for the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`. If the macro `\(language)hyphenmins` is already defined this command has no effect.

```

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

```

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

```

871 \def\set@hyphenmins#1#2{%
872 \lefthyphenmin#1\relax
873 \righthyphenmin#2\relax}

```

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

```

874 \ifx\ProvidesFile\@undefined
875 \def\ProvidesLanguage#1[#2 #3 #4]{%
876 \wlog{Language: #1 #4 #3 <#2>}%
877 }
878 \else
879 \def\ProvidesLanguage#1{%
880 \begingroup

```

```

881     \catcode`\ 10 %
882     \@makeother\/%
883     \@ifnextchar[%
884         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}]
885     \def\@provideslanguage#1[#2]{%
886         \wlog{Language: #1 #2}%
887         \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
888         \endgroup}
889 \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`.

```
890 \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.

```
891 \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’:

```

892 \providecommand\setlocale{\bbl@error{not-yet-available}}{}{}
893 \let\uselocale\setlocale
894 \let\locale\setlocale
895 \let\selectlocale\setlocale
896 \let\textlocale\setlocale
897 \let\textlanguage\setlocale
898 \let\languagegettext\setlocale

```

4.2 Errors

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

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

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

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

```

899 \edef\bbl@nulllanguage{\string\language=0}
900 \def\bbl@nocaption{\protect\bbl@nocaption@i}
901 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
902     \global\@namedef{#2}{\textbf{?#1?}}}%
903     \@nameuse{#2}%
904     \edef\bbl@tempa{#1}%
905     \bbl@sreplace\bbl@tempa{name}}}%
906     \bbl@warning{%
907         \@backslashchar#1 not set for '\language'. Please,\%
908         define it after the language has been loaded\%
909         (typically in the preamble) with:\%
910         \string\setlocalecaption{\language}{\bbl@tempa}{..}\%
911         Feel free to contribute on github.com/latex3/babel.\%
912         Reported}}
913 \def\bbl@tentative{\protect\bbl@tentative@i}
914 \def\bbl@tentative@i#1{%
915     \bbl@warning{%
916         Some functions for '#1' are tentative.\%
917         They might not work as expected and their behavior\%
918         could change in the future.\%
919         Reported}}
920 \def\@nolanerr#1{\bbl@error{undefined-language}{#1}}{}{}
921 \def\@nopatterns#1{%

```



```

922 \bbl@warning
923   {No hyphenation patterns were preloaded for\\%
924     the language '#1' into the format.\\%
925     Please, configure your TeX system to add them and\\%
926     rebuild the format. Now I will use the patterns\\%
927     preloaded for \bbl@nulllanguage\space instead}}
928 \let\bbl@usehooks@gobbletwo
929 \ifx\bbl@onlyswitch@empty\endinput\fi
930 % Here ended switch.def

```

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

```

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

```

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

```

969 \def\bbl@withactive#1#2{%
970   \begingroup
971   \lccode`~=#2\relax
972   \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`.

```

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

```

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

```

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

```

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

```

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

```

1014 \def\bb@evargs{,% <- don't delete this comma
1015 everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1016 adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1017 beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1018 hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1019 beforestart=0,language=2,begindocument=1}
1020 \ifx\NewHook\@undefined\else % Test for Plain (?)
1021 \def\bb@tempa#1=#2\@{\NewHook{babel/#1}}
1022 \bb@foreach\bb@evargs{\bb@tempa#1\@}
1023 \fi

```

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

```

1024 \bb@trace{Defining babelensure}
1025 \newcommand\babelensure[2][]{%
1026 \AddBabelHook{babel-ensure}{afterextras}{%
1027 \ifcase\bb@select@type
1028 \bb@cl{e}%
1029 \fi}%
1030 \beginingroup
1031 \let\bb@ens@include\@empty
1032 \let\bb@ens@exclude\@empty
1033 \def\bb@ens@fontenc{\relax}%
1034 \def\bb@tempb##1{%
1035 \ifx\@empty##1\else\noexpand##1\expandafter\bb@tempb\fi}%
1036 \edef\bb@tempa{\bb@tempb#1\@empty}%
1037 \def\bb@tempb##1=##2\@{\@namedef{bb@ens@##1}{##2}}%
1038 \bb@foreach\bb@tempa{\bb@tempb##1\@}%
1039 \def\bb@tempc{\bb@ensure}%
1040 \expandafter\bb@add\expandafter\bb@tempc\expandafter{%
1041 \expandafter{\bb@ens@include}}%
1042 \expandafter\bb@add\expandafter\bb@tempc\expandafter{%
1043 \expandafter{\bb@ens@exclude}}%
1044 \toks@\expandafter{\bb@tempc}%
1045 \bb@exp{%
1046 \endgroup
1047 \def<\bb@e@#2>{\the\toks@{\bb@ens@fontenc}}%
1048 \def\bb@ensure#1#2#3% 1: include 2: exclude 3: fontenc
1049 \def\bb@tempb##1% elt for (excluding) \bb@captionslist list
1050 \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1051 \edef##1{\noexpand\bb@nocaption
1052 {\bb@stripslash##1}{\language\bb@stripslash##1}}%
1053 \fi
1054 \ifx##1\@empty\else
1055 \in@{##1}{#2}%
1056 \ifin\@else
1057 \bb@ifunset{bb@ensure@\language}%
1058 {\bb@exp{%
1059 \\\DeclareRobustCommand<\bb@ensure@\language>[1]{%
1060 \\\foreignlanguage{\language}%
1061 {\ifx\relax#3\else
1062 \\\fontencoding{#3}\selectfont
1063 \fi
1064 #####1}}}%
1065 }%
1066 \toks@\expandafter{##1}%

```

```

1067     \edef##1{%
1068         \bbl@csarg\noexpand{ensure@\languagename}%
1069         {\the\toks@}}%
1070     \fi
1071     \expandafter\bbl@tempb
1072     \fi}%
1073 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1074 \def\bbl@tempa##1{% elt for include list
1075     \ifx##1\@empty\else
1076         \bbl@csarg\in@{ensure@\languagename\expandafter}\expandafter{##1}%
1077         \ifin@ \else
1078             \bbl@tempb##1\@empty
1079         \fi
1080     \expandafter\bbl@tempa
1081     \fi}%
1082 \bbl@tempa#1\@empty}
1083 \def\bbl@captionslist{%
1084     \prefacename\refname\abstractname\bibname\chaptername\appendixname
1085     \contentsname\listfigurename\listtablename\indexname\figurename
1086     \tablename\partname\enclname\ccname\headtoname\pagename\seename
1087     \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 at-sign. We make sure that it is a 'letter' during the processing of the file. We also save its name as the last called option, even if not loaded.

Another character that needs to have the correct category code during processing of language definition files is the equals sign, '=', because it is sometimes used in constructions with the `\let` primitive. Therefore we store its current catcode and restore it later on.

Now we check whether we should perhaps stop the processing of this file. To do this we first need to check whether the second argument that is passed to `\LdfInit` is a control sequence. We do that by looking at the first token after passing #2 through `string`. When it is equal to `\@backslashchar` we are dealing with a control sequence which we can compare with `\undefined`.

If so, we call `\ldf@quit` to set the main language, restore the category code of the @-sign and call `\endinput`

When #2 was *not* a control sequence we construct one and compare it with `\relax`.

Finally we check `\originalTeX`.

```

1088 \bbl@trace{Macros for setting language files up}
1089 \def\bbl@ldfinit{%
1090     \let\bbl@screset\@empty
1091     \let\BabelStrings\bbl@opt@string
1092     \let\BabelOptions\@empty
1093     \let\BabelLanguages\relax
1094     \ifx\originalTeX\undefined
1095         \let\originalTeX\@empty
1096     \else
1097         \originalTeX
1098     \fi}
1099 \def\LdfInit#1#2{%
1100     \chardef\atcatcode=\catcode`\@
1101     \catcode`\@=11\relax
1102     \chardef\eqcatcode=\catcode`=
1103     \catcode`\=12\relax
1104     \expandafter\if\expandafter\@backslashchar
1105         \expandafter\@car\string#2\@nil
1106     \ifx#2\undefined\else
1107         \ldf@quit{#1}%
1108     \fi

```

```

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

```

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

```

1115 \def\ldf@quit#1{%
1116   \expandafter\main@language\expandafter{#1}%
1117   \catcode`\@=\atcatcode \let\atcatcode\relax
1118   \catcode`\==\eqcatcode \let\eqcatcode\relax
1119   \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.

```

1120 \def\bbl@afterldf#1{%%^^A TODO. #1 is not used. Remove
1121   \bbl@afterlang
1122   \let\bbl@afterlang\relax
1123   \let\BabelModifiers\relax
1124   \let\bbl@screset\relax}%
1125 \def\ldf@finish#1{%
1126   \loadlocalcfg{#1}%
1127   \bbl@afterldf{#1}%
1128   \expandafter\main@language\expandafter{#1}%
1129   \catcode`\@=\atcatcode \let\atcatcode\relax
1130   \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 .

```

1131 \@onlypreamble\LdfInit
1132 \@onlypreamble\ldf@quit
1133 \@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.

```

1134 \def\main@language#1{%
1135   \def\bbl@main@language{#1}%
1136   \let\languagename\bbl@main@language
1137   \let\localename\bbl@main@language
1138   \let\mainlocalename\bbl@main@language
1139   \bbl@id@assign
1140   \bbl@patterns{\languagename}}

```

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

The code written to the aux file attempts to avoid errors if babel is removed from the document.

```

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

```

```

1152     \providecommand\babel@aux[2]{\global\let\babel@toc@gobbletwo}}%
1153     \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1154     \fi
1155     \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1156 </package | core>
1157 <*package>
1158     \ifx\bbl@normalsf@empty
1159         \ifnum\sfcodes\.=\@m
1160             \let\normalsfcodes\frenchspacing
1161         \else
1162             \let\normalsfcodes\nofrenchspacing
1163         \fi
1164     \else
1165         \let\normalsfcodes\bbl@normalsf
1166     \fi
1167 </package>
1168 <*package | core>
1169     \ifbbl@single % must go after the line above.
1170         \renewcommand\selectlanguage[1]{}%
1171         \renewcommand\foreignlanguage[2]{#2}%
1172         \global\let\babel@aux@gobbletwo % Also as flag
1173     \fi}
1174 </package | core>
1175 <*package>
1176 \AddToHook{begindocument/before}{%
1177     \let\bbl@normalsf\normalsfcodes
1178     \let\normalsfcodes\relax} % Hack, to delay the setting
1179 </package>%
1180 <*package | core>
1181 \ifcase\bbl@engine\or
1182     \AtBeginDocument{\pagedir\bodydir} %^^A TODO - a better place
1183 \fi

```

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

```

1184 \def\select@language@x#1{%
1185     \ifcase\bbl@select@type
1186         \bbl@ifsamestring\languagename{#1}{\select@language{#1}}%
1187     \else
1188         \select@language{#1}%
1189     \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 \LaTeX is used). It is used only at one place, namely when `\initiate@active@char` is called (which is ignored if the char has been made active before). Because `\@sanitize` can be undefined, we put the definition inside a conditional. Items are added to the lists without checking its existence or the original catcode. It does not hurt, but should be fixed. It's already done with `\nfss@catcodes`, added in 3.10.

```

1190 \bbl@trace{Shorhands}
1191 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1192     \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1193     \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\@makeother#1}}%
1194     \ifx\nfss@catcodes\undefined\else % TODO - same for above
1195         \begingroup
1196             \catcode`#1\active
1197             \nfss@catcodes
1198             \ifnum\catcode`#1=\active
1199                 \endgroup
1200                 \bbl@add\nfss@catcodes{\@makeother#1}%
1201             \else
1202                 \endgroup

```

```

1203     \fi
1204 \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.

```

1205 \def\bbl@remove@special#1{%
1206   \begingroup
1207   \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1208     \else\noexpand##1\noexpand##2\fi}%
1209   \def\do{\x\do}%
1210   \def\@makeother{\x\@makeother}%
1211   \edef\x{\endgroup
1212     \def\noexpand\dospecials{\dospecials}%
1213     \expandafter\ifx\csname @sanitize\endcsname\relax\else
1214       \def\noexpand\@sanitize{\@sanitize}%
1215     \fi}%
1216   \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).

```

1217 \def\bbl@active@def#1#2#3#4{%
1218   \@namedef{#3#1}{%
1219     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1220       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1221     \else
1222       \bbl@afterfi\csname#2@sh@#1\endcsname
1223     \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.

```

1224   \long\@namedef{#3@arg#1}##1{%
1225     \expandafter\ifx\csname#2@sh@#1@string##1\endcsname\relax
1226       \bbl@afterelse\csname#4#1\endcsname##1%
1227     \else
1228       \bbl@afterfi\csname#2@sh@#1@string##1\endcsname
1229     \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.

```

1230 \def\initiate@active@char#1{%
1231   \bbl@ifunset{active@char\string#1}%
1232   {\bbl@withactive
1233     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1234   {}}

```

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).

```

1235 \def\@initiate@active@char#1#2#3{%
1236   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1237   \ifx#1\@undefined
1238     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1239   \else
1240     \bbl@csarg\let{oridef@#2}#1%
1241     \bbl@csarg\edef{oridef@#2}{%
1242       \let\noexpand#1%
1243       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1244   \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*").

```

1245   \ifx#1#3\relax
1246     \expandafter\let\csname normal@char#2\endcsname#3%
1247   \else
1248     \bbl@info{Making #2 an active character}%
1249     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1250     \@namedef{normal@char#2}{%
1251       \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
1252   \else
1253     \@namedef{normal@char#2}{#3}%
1254   \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).

```

1255   \bbl@restoreactive{#2}%
1256   \AtBeginDocument{%
1257     \catcode`#2\active
1258     \if@filesw
1259       \immediate\write\@mainaux{\catcode`\string#2\active}%
1260     \fi}%
1261   \expandafter\bbl@add@special\csname#2\endcsname
1262   \catcode`#2\active
1263   \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⟩`).

```

1264   \let\bbl@tempa\@firstoftwo
1265   \if\string^#2%
1266     \def\bbl@tempa{\noexpand\textormath}%
1267   \else
1268     \ifx\bbl@mathnormal\@undefined\else
1269       \let\bbl@tempa\bbl@mathnormal
1270     \fi
1271   \fi
1272   \expandafter\edef\csname active@char#2\endcsname{%
1273     \bbl@tempa
1274     {\noexpand\if@safe@actives
1275       \noexpand\expandafter
1276       \expandafter\noexpand\csname normal@char#2\endcsname
1277     \noexpand\else
1278       \noexpand\expandafter
1279       \expandafter\noexpand\csname bbl@doactive#2\endcsname

```



```

1280     \noexpand\fi}%
1281     {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1282     \bbl@csarg\edef{doactive#2}{%
1283     \expandafter\noexpand\csname user@active#2\endcsname}%

```

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

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

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

```

1284     \bbl@csarg\edef{active@#2}{%
1285     \noexpand\active@prefix\noexpand#1%
1286     \expandafter\noexpand\csname active@char#2\endcsname}%
1287     \bbl@csarg\edef{normal@#2}{%
1288     \noexpand\active@prefix\noexpand#1%
1289     \expandafter\noexpand\csname normal@char#2\endcsname}%
1290     \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.

```

1291     \bbl@active@def#2\user@group{user@active}{language@active}%
1292     \bbl@active@def#2\language@group{language@active}{system@active}%
1293     \bbl@active@def#2\system@group{system@active}{normal@char}%

```

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

```

1294     \expandafter\edef\csname\user@group @sh@#2@@\endcsname
1295     {\expandafter\noexpand\csname normal@char#2\endcsname}%
1296     \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
1297     {\expandafter\noexpand\csname user@active#2\endcsname}%

```

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

```

1298     \if\string'#2%
1299     \let\prim@s\bbl@prim@s
1300     \let\active@math@prime#1%
1301     \fi
1302     \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}

```

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

```

1303 <<(*More package options)>> ≡
1304 \DeclareOption{math=active}{}
1305 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1306 <</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.

```

1307 \ifpackagewith{babel}{KeepShorthandsActive}%
1308 {\let\bbl@restoreactive@gobble}%
1309 {\def\bbl@restoreactive#1{%
1310     \bbl@exp{%
1311         \\AfterBabelLanguage\\CurrentOption
1312         {\catcode`#1=\the\catcode`#1\relax}%
1313         \\AtEndOfPackage
1314         {\catcode`#1=\the\catcode`#1\relax}}}%
1315 \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.

```

1316 \def\bbl@sh@select#1#2{%
1317   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1318   \bbl@afterelse\bbl@scndcs
1319   \else
1320   \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1321   \fi}

```

`\active@prefix` The command `\active@prefix` which is used in the expansion of active characters has a function similar to `\OT1-cmd` in that it `\protects` the active character whenever `\protect` is *not* `\@typeset@protect`. The `\@gobble` is needed to remove a token such as `\activechar:` (when the double colon was the active character to be dealt with). There are two definitions, depending of `\ifincsname` is available. If there is, the expansion will be more robust.

```

1322 \begingroup
1323 \bbl@ifunset{ifincsname}%^^A Ugly. Correct? Only Plain?
1324 {\gdef\active@prefix#1{%
1325   \ifx\protect\@typeset@protect
1326   \else
1327   \ifx\protect\@unexpandable@protect
1328   \noexpand#1%
1329   \else
1330   \protect#1%
1331   \fi
1332   \expandafter\@gobble
1333   \fi}}
1334 {\gdef\active@prefix#1{%
1335   \ifincsname
1336   \string#1%
1337   \expandafter\@gobble
1338   \else
1339   \ifx\protect\@typeset@protect
1340   \else
1341   \ifx\protect\@unexpandable@protect
1342   \noexpand#1%
1343   \else
1344   \protect#1%
1345   \fi
1346   \expandafter\expandafter\expandafter\@gobble
1347   \fi
1348   \fi}}
1349 \endgroup

```

`\if@safe@actives` In some circumstances it is necessary to be able to reset the shorthand to its ‘normal’ value (usually the character with catcode ‘other’) on the fly. For this purpose the switch `@safe@actives` is available. The setting of this switch should be checked in the first level expansion of `\active@char<char>`. When this expansion mode is active (with `\@safe@activestrue`), something like `"13"13` becomes `"12"12` in an `\edef` (in other words, shorthands are `\string'ed`). This contrasts with `\protected@edef`, where catcodes are always left unchanged. Once converted, they can be used safely even after this expansion mode is deactivated (with `\@safe@activesfalse`).

```

1350 \newif\if@safe@actives
1351 \@safe@activesfalse

```

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

```

1352 \def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}

```

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

```

1353 \chardef\bbl@activated\z@
1354 \def\bbl@activate#1{%
1355   \chardef\bbl@activated\@ne
1356   \bbl@withactive{\expandafter\let\expandafter}#1%
1357   \csname bbl@active@\string#1\endcsname}
1358 \def\bbl@deactivate#1{%
1359   \chardef\bbl@activated\tw@
1360   \bbl@withactive{\expandafter\let\expandafter}#1%
1361   \csname bbl@normal@\string#1\endcsname}

```

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

```

\bbl@scndcs 1362 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1363 \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.

```

1364 \def\babel@texpdf#1#2#3#4{%
1365   \ifx\texorpdfstring\undefined
1366     \textormath{#1}{#3}%
1367   \else
1368     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1369     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1370   \fi}
1371 %
1372 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1373 \def\@decl@short#1#2#3\@nil#4{%
1374   \def\bbl@tempa{#3}%
1375   \ifx\bbl@tempa\empty
1376     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1377     \bbl@ifunset{#1@sh@\string#2@}{}%
1378     {\def\bbl@tempa{#4}%
1379      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1380      \else
1381        \bbl@info
1382        {Redefining #1 shorthand \string#2\%
1383         in language \CurrentOption}%
1384      \fi}%
1385     \@namedef{#1@sh@\string#2@}{#4}%
1386   \else
1387     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1388     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1389     {\def\bbl@tempa{#4}%
1390      \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1391      \else
1392        \bbl@info
1393        {Redefining #1 shorthand \string#2\string#3\%
1394         in language \CurrentOption}%
1395      \fi}%
1396     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1397   \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.

```
1398 \def\textormath{%
1399   \ifmmode
1400     \expandafter\@secondoftwo
1401   \else
1402     \expandafter\@firstoftwo
1403   \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’.

```
1404 \def\user@group{user}
1405 \def\language@group{english} %^^A I don't like defaults
1406 \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.

```
1407 \def\useshorthands{%
1408   \ifstar\bb@usesh@s{\bb@usesh@x{}}
1409 \def\bb@usesh@s#1{%
1410   \bb@usesh@x
1411   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb@activate{#1}}}%
1412   {#1}}
1413 \def\bb@usesh@x#1#2{%
1414   \bb@ifshorthand{#2}%
1415   {\def\user@group{user}%
1416     \initiate@active@char{#2}%
1417     #1%
1418     \bb@activate{#2}}%
1419   {\bb@error{shorthand-is-off}{#2}{}}}
```

`\defineshorthand` Currently we only support two groups of user level shorthands, named internally `user` and `user@<language>` (language-dependent user shorthands). By default, only the first one is taken into account, but if the former is also used (in the optional argument of `\defineshorthand`) a new level is inserted for it (`user@generic`, done by `\bb@set@user@generic`); we make also sure `{}` and `\protect` are taken into account in this new top level.

```
1420 \def\user@language@group{user@language@group}
1421 \def\bb@set@user@generic#1#2{%
1422   \bb@ifunset{user@generic@active#1}%
1423   {\bb@active@def#1\user@language@group{user@active}{user@generic@active}%
1424     \bb@active@def#1\user@group{user@generic@active}{language@active}%
1425     \expandafter\edef\csname#2@sh@#1@\endcsname{%
1426       \expandafter\noexpand\csname normal@char#1\endcsname}%
1427     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1428       \expandafter\noexpand\csname user@active#1\endcsname}}%
1429   \@empty}
1430 \newcommand\defineshorthand[3][user]{%
1431   \edef\bb@tempa{\zap@space#1 \@empty}%
1432   \bb@for\bb@tempb\bb@tempa{%
1433     \if*\expandafter\@car\bb@tempb\@nil
1434     \edef\bb@tempb{user@\expandafter\@gobble\bb@tempb}%
1435     \@expandtwoargs
1436     \bb@set@user@generic{\expandafter\string\@car#2\@nil}\bb@tempb
1437   \fi
1438   \declare@shorthand{\bb@tempb}{#2}{#3}}}
```

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

```
1439 \def\languageshorthands#1{\def\language@group{#1}}
```

`\aliasshorthand` *Deprecated*. First the new shorthand needs to be initialized. Then, we define the new shorthand in terms of the original one, but note with `\aliasshorthands{"}/}` is `\active@prefix /\active@char/`, so we still need to let the latter to `\active@char`.

```

1440 \def\aliasshorthand#1#2{%
1441   \bbl@ifshorthand{#2}%
1442     {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1443       \ifx\document\@notprerr
1444         \@notshorthand{#2}%
1445       \else
1446         \initiate@active@char{#2}%
1447         \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1448         \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1449         \bbl@activate{#2}%
1450       \fi
1451     \fi}%
1452   {\bbl@error{shorthand-is-off}{#2}{}}}
```

`\@notshorthand`

```

1453 \def\@notshorthand#1{\bbl@error{not-a-shorthand}{#1}{}}
```

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

```

1454 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1455 \DeclareRobustCommand*\shorthandoff{%
1456   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1457 \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.

```

1458 \def\bbl@switch@sh#1#2{%
1459   \ifx#2\@nnil\else
1460     \bbl@ifunset{bbl@active@\string#2}%
1461     {\bbl@error{not-a-shorthand-b}{#2}{}}%
1462     {\ifcase#1%   off, on, off*
1463       \catcode`#2\relax
1464     \or
1465       \catcode`#2\active
1466       \bbl@ifunset{bbl@shdef@\string#2}%
1467       {}%
1468       {\bbl@withactive{\expandafter\let\expandafter}#2%
1469         \csname bbl@shdef@\string#2\endcsname
1470         \bbl@csarg\let{shdef@\string#2}\relax}%
1471       \ifcase\bbl@activated\or
1472         \bbl@activate{#2}%
1473       \else
1474         \bbl@deactivate{#2}%
1475       \fi
1476     \or
1477       \bbl@ifunset{bbl@shdef@\string#2}%
1478       {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1479       {}%
1480       \csname bbl@oricat@\string#2\endcsname
1481       \csname bbl@oridef@\string#2\endcsname
1482     \fi}%
1483   \bbl@afterfi\bbl@switch@sh#1%
1484   \fi}
```

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

```

1485 \def\babelshorthand{\active@prefix\babelshorthand\bb@putsh}
1486 \def\bb@putsh#1{%
1487   \bb@ifunset{bb@active@\string#1}%
1488     {\bb@putsh@i#1@empty@nnil}%
1489     {\csname bb@active@\string#1\endcsname}}
1490 \def\bb@putsh@i#1#2@nnil{%
1491   \csname\language@group @sh@\string#1@%
1492     \ifx@empty#2\else\string#2@fi\endcsname}
1493 %
1494 \ifx\bb@opt@shorthands@nnil\else
1495   \let\bb@s@initiate@active@char\initiate@active@char
1496   \def\initiate@active@char#1{%
1497     \bb@ifshorthand{#1}{\bb@s@initiate@active@char{#1}}{}}
1498   \let\bb@s@switch@sh\bb@switch@sh
1499   \def\bb@switch@sh#1#2{%
1500     \ifx#2@nnil\else
1501       \bb@afterfi
1502       \bb@ifshorthand{#2}{\bb@s@switch@sh#1{#2}}{\bb@switch@sh#1}%
1503       \fi}
1504   \let\bb@s@activate\bb@activate
1505   \def\bb@activate#1{%
1506     \bb@ifshorthand{#1}{\bb@s@activate{#1}}{}}
1507   \let\bb@s@deactivate\bb@deactivate
1508   \def\bb@deactivate#1{%
1509     \bb@ifshorthand{#1}{\bb@s@deactivate{#1}}{}}
1510 \fi

```

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

```

1511 \newcommand\ifbabelshorthand[3]{\bb@ifunset{bb@active@\string#1}{#3}{#2}}

```

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

```

1512 \def\bb@prim@s{%
1513   \prime\futurelet\@let@token\bb@pr@m@s}
1514 \def\bb@if@primes#1#2{%
1515   \ifx#1\@let@token
1516     \expandafter\@firstoftwo
1517   \else\ifx#2\@let@token
1518     \bb@afterelse\expandafter\@firstoftwo
1519   \else
1520     \bb@afterfi\expandafter\@secondoftwo
1521   \fi\fi}
1522 \begingroup
1523 \catcode\^=7 \catcode\*=\active \lccode\^=\^
1524 \catcode\'=12 \catcode\"=\active \lccode\"=\`
1525 \lowercase{%
1526   \gdef\bb@pr@m@s{%
1527     \bb@if@primes" '%
1528     \pr@@s
1529     {\bb@if@primes*\^ \pr@@t\egroup}}
1530 \endgroup

```

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

```

1531 \initiate@active@char{~}
1532 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1533 \bbl@activate{~}

```

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

```

1534 \expandafter\def\csname OT1dqpos\endcsname{127}
1535 \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

```

1536 \ifx\f@encoding\@undefined
1537 \def\f@encoding{OT1}
1538 \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.

```

1539 \bbl@trace{Language attributes}
1540 \newcommand\languageattribute[2]{%
1541 \def\bbl@tempc{#1}%
1542 \bbl@fixname\bbl@tempc
1543 \bbl@iflanguage\bbl@tempc{%
1544 \bbl@vforeach{#2}{%

```

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

```

1545 \ifx\bbl@known@attribs\@undefined
1546 \in@false
1547 \else
1548 \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
1549 \fi
1550 \ifin@
1551 \bbl@warning{%
1552 You have more than once selected the attribute '##1'\%
1553 for language #1. Reported}%
1554 \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.

```

1555 \bbl@exp{%
1556 \\\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
1557 \edef\bbl@tempa{\bbl@tempc-##1}%
1558 \expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes%
1559 {\csname\bbl@tempc @attr##1\endcsname}%
1560 {\@attrerr{\bbl@tempc}{##1}}%
1561 \fi}}
1562 \@onlypreamble\languageattribute

```

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

```

1563 \newcommand*\@attrerr}[2]{%
1564 \bbl@error{unknown-attribute}{#1}{#2}{}}

```

`\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}`.

```

1565 \def\babel@declare@ttribute#1#2#3{%
1566   \babel@xin@{,#2,}{,\BabelModifiers,}%
1567   \ifin@
1568     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1569   \fi
1570   \babel@add@list\babel@attributes{#1-#2}%
1571   \expandafter\def\csname#1@attr@#2\endcsname{#3}}

```

`\babel@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.

```

1572 \def\babel@ifattributeset#1#2#3#4{%
1573   \ifx\babel@known@attrs\@undefined
1574     \in@false
1575   \else
1576     \babel@xin@{,#1-#2,}{,\babel@known@attrs,}%
1577   \fi
1578   \ifin@
1579     \babel@afterelse#3%
1580   \else
1581     \babel@afterfi#4%
1582   \fi}

```

`\babel@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.

```

1583 \def\babel@ifknown@ttrib#1#2{%
1584   \let\babel@tempa\@secondoftwo
1585   \babel@loopx\babel@tempb{#2}{%
1586     \expandafter\in\expandafter{\expandafter,\babel@tempb,}{,#1,}%
1587   \ifin@
1588     \let\babel@tempa\@firstoftwo
1589   \else
1590   \fi}%
1591   \babel@tempa}

```

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

```

1592 \def\babel@clear@ttribs{%
1593   \ifx\babel@attributes\@undefined\else
1594     \babel@loopx\babel@tempa{\babel@attributes}{%
1595       \expandafter\babel@clear@ttrib\babel@tempa.}%
1596     \let\babel@attributes\@undefined
1597   \fi}
1598 \def\babel@clear@ttrib#1-#2.{%
1599   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1600 \AtBeginDocument{\babel@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`

```
1601 \bbl@trace{Macros for saving definitions}
1602 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1603 \newcount\babel@savecnt
1604 \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.

```
1605 \def\babel@save#1{%
1606   \def\bbl@tempa{,{, #1,}}% Clumsy, for Plain
1607   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1608     \expandafter{\expandafter, \bbl@savextras,}}%
1609   \expandafter\in@\bbl@tempa
1610   \ifin@ \else
1611     \bbl@add\bbl@savextras{, #1,}%
1612     \bbl@carg\let{babel@\number\babel@savecnt}#1\relax
1613     \toks@\expandafter{\originalTeX\let#1=}%
1614     \bbl@exp{%
1615       \def\\originalTeX{\the\toks@<babel@\number\babel@savecnt>\relax}}%
1616     \advance\babel@savecnt@ne
1617   \fi}
1618 \def\babel@savevariable#1{%
1619   \toks@\expandafter{\originalTeX #1=}%
1620   \bbl@exp{\def\\originalTeX{\the\toks@ \the#1\relax}}}
```

`\bbl@frenchspacing` Some languages need to have `\frenchspacing` in effect. Others don't want that. The command `\bbl@nonfrenchspacing` switches it off if necessary. A more refined way to switch the catcodes is done with ini files. Here an auxiliary macro is defined, but the main part is in `\babelprovide`. This new method should be ideally the default one.

```
1621 \def\bbl@frenchspacing{%
1622   \ifnum\the\sfcode`.=\@m
1623     \let\bbl@nonfrenchspacing\relax
1624   \else
1625     \frenchspacing
1626     \let\bbl@nonfrenchspacing\nonfrenchspacing
1627   \fi}
1628 \let\bbl@nonfrenchspacing\nonfrenchspacing
1629 \let\bbl@elt\relax
1630 \edef\bbl@fs@chars{%
1631   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1632   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1633   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1634 \def\bbl@pre@fs{%
1635   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1636   \edef\bbl@save@sfcodes{\bbl@fs@chars}%
1637 \def\bbl@post@fs{%
1638   \bbl@save@sfcodes
1639   \edef\bbl@tempa{\bbl@cl{frspc}}%
1640   \edef\bbl@tempa{\expandafter\@car\bbl@tempa@nil}%
1641   \if u\bbl@tempa % do nothing
1642   \else\if n\bbl@tempa % non french
1643     \def\bbl@elt##1##2##3{%
1644       \ifnum\sfcode`##1=##2\relax
1645       \babel@savevariable{\sfcode`##1}%
```

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

```

1646     \sfcode`##1=##3\relax
1647     \fi}%
1648     \bbl@fs@chars
1649 \else\if y\bbl@tempa      % french
1650     \def\bbl@elt##1##2##3{%
1651         \ifnum\sfcode`##1=##3\relax
1652             \babel@savevariable{\sfcode`##1}%
1653             \sfcode`##1=##2\relax
1654         \fi}%
1655     \bbl@fs@chars
1656     \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.

```

1657 \bbl@trace{Short tags}
1658 \def\babeltags#1{%
1659     \edef\bbl@tempa{\zap@space#1 \@empty}%
1660     \def\bbl@tempb##1=##2\@{%
1661         \edef\bbl@tempc{%
1662             \noexpand\newcommand
1663             \expandafter\noexpand\csname ##1\endcsname{%
1664                 \noexpand\protect
1665                 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1666             \noexpand\newcommand
1667             \expandafter\noexpand\csname text##1\endcsname{%
1668                 \noexpand\foreignlanguage{##2}}
1669         \bbl@tempc}%
1670     \bbl@for\bbl@tempa\bbl@tempa{%
1671         \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<language>` for language ones. See `\bbl@patterns` above for further details. We make sure there is a space between words when multiple commands are used.

```

1672 \bbl@trace{Hyphens}
1673 \@onlypreamble\babelhyphenation
1674 \AtEndOfPackage{%
1675     \newcommand\babelhyphenation[2][\@empty]{%
1676         \ifx\bbl@hyphenation@relax
1677             \let\bbl@hyphenation@\@empty
1678         \fi
1679         \ifx\bbl@hyphlist\@empty\else
1680             \bbl@warning{%
1681                 You must not intermingle \string\selectlanguage\space and\\%
1682                 \string\babelhyphenation\space or some exceptions will not\\%
1683                 be taken into account. Reported}%
1684             \fi
1685         \ifx\@empty#1%
1686             \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1687         \else
1688             \bbl@vforeach{#1}{%
1689                 \def\bbl@tempa{##1}%
1690                 \bbl@fixname\bbl@tempa
1691                 \bbl@iflanguage\bbl@tempa{%
1692                     \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1693                         \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1694                         {}%
1695                     {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%

```

```

1696         #2}}}%
1697     \fi}}

```

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

```

1698 \ifx\NewDocumentCommand\undefined\else
1699   \NewDocumentCommand\babelhyphenmins{sommo}{%
1700     \IfNoValueTF{#2}%
1701     {\protected@edef\bbL@hyphenmins@{\set@hyphenmins{#3}{#4}}%
1702     \IfValueT{#5}{%
1703       \protected@edef\bbL@hyphenatmin@{\hyphenationmin=#5\relax}}%
1704     \IfBooleanT{#1}{%
1705       \lefthyphenmin=#3\relax
1706       \righthyphenmin=#4\relax
1707       \IfValueT{#5}{\hyphenationmin=#5\relax}}}%
1708   {\edef\bbL@tempb{\zap@space#2 \@empty}%
1709   \bbL@for\bbL@tempa\bbL@tempb{%
1710     \namedef{bbL@hyphenmins@bbL@tempa}{\set@hyphenmins{#3}{#4}}%
1711     \IfValueT{#5}{%
1712       \namedef{bbL@hyphenatmin@bbL@tempa}{\hyphenationmin=#5\relax}}}%
1713   \IfBooleanT{#1}{\bbL@error{hyphenmins-args}{}}}}
1714 \fi

```

`\bbL@allowhyphens` This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak \hskip 0pt plus 0pt3`.

```

1715 \def\bbL@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1716 \def\bbL@t@one{T1}
1717 \def\allowhyphens{\ifx\c@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`.

```

1718 \newcommand\babelnulLhyphen{\char\hyphenchar\font}
1719 \def\babelhyphen{\active@prefix\babelhyphen\bbL@hyphen}
1720 \def\bbL@hyphen{%
1721   \ifstar{\bbL@hyphen@i @}{\bbL@hyphen@i \@empty}}
1722 \def\bbL@hyphen@i#1#2{%
1723   \bbL@ifunset{bbL@hy@#1#2\@empty}%
1724   {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1725   {\csname bbl@hy@#1#2\@empty\endcsname}}

```

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

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

```

1726 \def\bbL@usehyphen#1{%
1727   \leavevmode
1728   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1729   \nobreak\hskip\z@skip}
1730 \def\bbL@@usehyphen#1{%
1731   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char:

```

1732 \def\bbL@hyphenchar{%
1733   \ifnum\hyphenchar\font=\m@ne
1734     \babelnulLhyphen
1735   \else
1736     \char\hyphenchar\font
1737   \fi}

```

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

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@nbreak` is redundant.

```

1738 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1739 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1740 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1741 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1742 \def\bbl@hy@nbreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1743 \def\bbl@hy@@nbreak{\mbox{\bbl@hyphenchar}}
1744 \def\bbl@hy@repeat{%
1745   \bbl@usehyphen{%
1746     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1747 \def\bbl@hy@@repeat{%
1748   \bbl@usehyphen{%
1749     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1750 \def\bbl@hy@empty{\hskip\z@skip}
1751 \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.

```

1752 \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.

```

1753 \bbl@trace{Multiencoding strings}
1754 \def\bbl@tglobal#1{\global\let#1#1}

```

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

```

1755 << *More package options >> ≡
1756 \DeclareOption{nocase}{}
1757 << /More package options >>

```

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

```

1758 << *More package options >> ≡
1759 \let\bbl@opt@strings\@nnil % accept strings=value
1760 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1761 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1762 \def\BabelStringsDefault{generic}
1763 << /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.

```

1764 \@onlypreamble\StartBabelCommands
1765 \def\StartBabelCommands{%
1766   \begingroup
1767   \@tempcnta="7F
1768   \def\bbl@tempa{%
1769     \ifnum\@tempcnta>"FF\else
1770       \catcode\@tempcnta=11
1771       \advance\@tempcnta\@ne
1772       \expandafter\bbl@tempa
1773     \fi}%
1774   \bbl@tempa
1775   <@Macros local to BabelCommands@>
1776   \def\bbl@provstring##1##2{%
1777     \providecommand##1{##2}%

```

```

1778 \bbl@tglobal##1}%
1779 \global\let\bbl@scafter\@empty
1780 \let\StartBabelCommands\bbl@startcmds
1781 \ifx\BabelLanguages\relax
1782 \let\BabelLanguages\CurrentOption
1783 \fi
1784 \begingroup
1785 \let\bbl@screaset\@nnil % local flag - disable 1st stopcommands
1786 \StartBabelCommands}
1787 \def\bbl@startcmds{%
1788 \ifx\bbl@screaset\@nnil\else
1789 \bbl@usehooks{stopcommands}{}%
1790 \fi
1791 \endgroup
1792 \begingroup
1793 \@ifstar
1794 {\ifx\bbl@opt@strings\@nnil
1795 \let\bbl@opt@strings\BabelStringsDefault
1796 \fi
1797 \bbl@startcmds@i}%
1798 \bbl@startcmds@i}
1799 \def\bbl@startcmds@i#1#2{%
1800 \edef\bbl@L{\zap@space#1 \@empty}%
1801 \edef\bbl@G{\zap@space#2 \@empty}%
1802 \bbl@startcmds@ii}
1803 \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.

```

1804 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1805 \let\SetString\@gobbletwo
1806 \let\bbl@stringdef\@gobbletwo
1807 \let\AfterBabelCommands\@gobble
1808 \ifx\@empty#1%
1809 \def\bbl@sc@label{generic}%
1810 \def\bbl@encstring##1##2{%
1811 \ProvideTextCommandDefault##1{##2}%
1812 \bbl@tglobal##1%
1813 \expandafter\bbl@tglobal\cname\string?\string##1\endcname}%
1814 \let\bbl@sctest\in@true
1815 \else
1816 \let\bbl@sc@charset\space % <- zapped below
1817 \let\bbl@sc@fontenc\space % <- " "
1818 \def\bbl@tempa##1=##2\@nil{%
1819 \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1820 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1821 \def\bbl@tempa##1 ##2{% space -> comma
1822 ##1%
1823 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1824 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1825 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1826 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1827 \def\bbl@encstring##1##2{%
1828 \bbl@foreach\bbl@sc@fontenc{%
1829 \bbl@ifunset{T@###1}%
1830 }%

```

```

1831     {\ProvideTextCommand###1{###1}{##2}%
1832     \bbl@tglobal##1%
1833     \expandafter
1834     \bbl@tglobal\csname###1\string##1\endcsname}}}%
1835 \def\bbl@sctest{%
1836     \bbl@xin@{\, \bbl@opt@strings,}{, \bbl@sc@label, \bbl@sc@fontenc,}}%
1837 \fi
1838 \ifx\bbl@opt@strings\@nnil      % ie, no strings key -> defaults
1839 \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
1840     \let\AfterBabelCommands\bbl@aftercmds
1841     \let\SetString\bbl@setstring
1842     \let\bbl@stringdef\bbl@encstring
1843 \else      % ie, strings=value
1844 \bbl@sctest
1845 \ifin@
1846     \let\AfterBabelCommands\bbl@aftercmds
1847     \let\SetString\bbl@setstring
1848     \let\bbl@stringdef\bbl@provstring
1849 \fi\fi\fi
1850 \bbl@scswitch
1851 \ifx\bbl@G\@empty
1852     \def\SetString##1##2{%
1853         \bbl@error{missing-group}{##1}{}}}%
1854 \fi
1855 \ifx\@empty#1%
1856     \bbl@usehooks{defaultcommands}{}%
1857 \else
1858     \@expandtwoargs
1859     \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1860 \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).

```

1861 \def\bbl@forlang#1#2{%
1862     \bbl@for#1\bbl@L{%
1863         \bbl@xin@{, #1,}{, \BabelLanguages,}%
1864         \ifin@#2\relax\fi}}
1865 \def\bbl@scswitch{%
1866     \bbl@forlang\bbl@tempa{%
1867         \ifx\bbl@G\@empty\else
1868             \ifx\SetString\@gobbletwo\else
1869                 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1870                 \bbl@xin@{\, \bbl@GL,}{, \bbl@screset,}%
1871                 \ifin@\else
1872                     \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1873                 \xdef\bbl@screset{\bbl@screset, \bbl@GL}%
1874             \fi
1875         \fi
1876     \fi}}
1877 \AtEndOfPackage{%
1878     \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{#2}}}%
1879     \let\bbl@scswitch\relax}
1880 \onlypreamble\EndBabelCommands
1881 \def\EndBabelCommands{%
1882     \bbl@usehooks{stopcommands}{}%
1883     \endgroup
1884     \endgroup

```

```
1885 \bbl@scafter}
1886 \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.

```
1887 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1888 \bbl@forlang\bbl@tempa{%
1889 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1890 \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1891 {\bbl@exp{%
1892 \global\\bbl@add\<\bbl@G\bbl@tempa>{\\bbl@scset\\#1\<\bbl@LC>}}}%
1893 }%
1894 \def\BabelString{#2}%
1895 \bbl@usehooks{stringprocess}{}%
1896 \expandafter\bbl@stringdef
1897 \csname\bbl@LC\expandafter\endcsname\expandafter{\BabelString}}
```

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

```
1898 \def\bbl@scset#1#2{\def#1{#2}}
```

Define \SetStringLoop, which is actually set inside \StartBabelCommands. The current definition is somewhat complicated because we need a count, but \count@ is not under our control (remember \SetString may call hooks). Instead of defining a dedicated count, we just “pre-expand” its value.

```
1899 <<{*Macros local to BabelCommands}>> ≡
1900 \def\SetStringLoop###1##2{%
1901 \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1902 \count@\z@
1903 \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1904 \advance\count@\@ne
1905 \toks@\expandafter{\bbl@tempa}%
1906 \bbl@exp{%
1907 \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1908 \count@=\the\count@\relax}}}%
1909 <</Macros local to BabelCommands>>
```

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

```
1910 \def\bbl@aftercmds#1{%
1911 \toks@\expandafter{\bbl@scafter#1}%
1912 \xdef\bbl@scafter{\the\toks@}}
```

Case mapping The command \SetCase is deprecated. Currently it consists in a definition with a hack just for backward compatibility in the macro mapping.

```
1913 <<{*Macros local to BabelCommands}>> ≡
1914 \newcommand\SetCase[3][]{%
1915 \def\bbl@tempa####1####2{%
1916 \ifx####1\@empty\else
1917 \bbl@carg\bbl@add{extras\CurrentOption}{%
1918 \bbl@carg\babel@save{c__text_uppercase\_string####1_tl}%
1919 \bbl@carg\def{c__text_uppercase\_string####1_tl}{####2}%
1920 \bbl@carg\babel@save{c__text_lowercase\_string####2_tl}%
1921 \bbl@carg\def{c__text_lowercase\_string####2_tl}{####1}}%
1922 \expandafter\bbl@tempa
1923 \fi}%
1924 \bbl@tempa##1\@empty\@empty
1925 \bbl@carg\bbl@tglobal{extras\CurrentOption}}%
1926 <</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.

```

1927 <<{*Macros local to BabelCommands}>> ≡
1928 \newcommand\SetHyphenMap[1]{%
1929   \bbl@forlang\bbl@tempa{%
1930     \expandafter\bbl@stringdef
1931     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1932 <</Macros local to BabelCommands>>

```

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

```

1933 \newcommand\BabelLower[2]{% one to one.
1934   \ifnum\lccode#1=#2\else
1935     \babel@savevariable{\lccode#1}%
1936     \lccode#1=#2\relax
1937   \fi}
1938 \newcommand\BabelLowerMM[4]{% many-to-many
1939   \@tempcnta=#1\relax
1940   \@tempcntb=#4\relax
1941   \def\bbl@tempa{%
1942     \ifnum\@tempcnta>#2\else
1943       \expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1944       \advance\@tempcnta#3\relax
1945       \advance\@tempcntb#3\relax
1946       \expandafter\bbl@tempa
1947     \fi}%
1948   \bbl@tempa}
1949 \newcommand\BabelLowerM0[4]{% many-to-one
1950   \@tempcnta=#1\relax
1951   \def\bbl@tempa{%
1952     \ifnum\@tempcnta>#2\else
1953       \expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1954       \advance\@tempcnta#3
1955       \expandafter\bbl@tempa
1956     \fi}%
1957   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1958 <<{*More package options}>> ≡
1959 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1960 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap@ne}
1961 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1962 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1963 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1964 <</More package options>>

```

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

```

1965 \AtEndOfPackage{%
1966   \ifx\bbl@opt@hyphenmap\undefined
1967     \bbl@xin@{,}{\bbl@language@opts}%
1968     \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
1969   \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.

```

1970 \newcommand\setlocalecaption{%%^^A Catch typos.
1971   \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
1972 \def\bbl@setcaption@x#1#2#3{% language caption-name string
1973   \bbl@trim@def\bbl@tempa{#2}%
1974   \bbl@xin@{.template}{\bbl@tempa}%
1975   \ifin@
1976     \bbl@ini@captions@template{#3}{#1}%

```



```

1977 \else
1978   \edef\bbbl@tempd{%
1979     \expandafter\expandafter\expandafter
1980     \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1981   \bbbl@xin@
1982     \expandafter\string\csname #2name\endcsname}%
1983     {\bbbl@tempd}%
1984   \ifin@ % Renew caption
1985     \bbbl@xin@{\string\bbbl@scset}{\bbbl@tempd}%
1986     \ifin@
1987       \bbbl@exp{%
1988         \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
1989         {\bbbl@scset\<#2name>\<#1#2name>}%
1990         {}}%
1991     \else % Old way converts to new way
1992       \bbbl@ifunset{#1#2name}%
1993         {\bbbl@exp{%
1994           \\bbbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
1995           \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
1996           {\def\<#2name>{\<#1#2name>}}%
1997           {}}}%
1998         {}%
1999     \fi
2000   \else
2001     \bbbl@xin@{\string\bbbl@scset}{\bbbl@tempd}% New
2002     \ifin@ % New way
2003       \bbbl@exp{%
2004         \\bbbl@add\<captions#1>{\bbbl@scset\<#2name>\<#1#2name>}%
2005         \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2006         {\bbbl@scset\<#2name>\<#1#2name>}%
2007         {}}%
2008     \else % Old way, but defined in the new way
2009       \bbbl@exp{%
2010         \\bbbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2011         \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2012         {\def\<#2name>{\<#1#2name>}}%
2013         {}}%
2014       \fi%
2015     \fi
2016     \@namedef{#1#2name}{#3}%
2017     \toks@\expandafter{\bbbl@captionslist}%
2018     \bbbl@exp{\in@{\<#2name>}{\the\toks@}}%
2019     \ifin@ \else
2020       \bbbl@exp{\bbbl@add\bbbl@captionslist{\<#2name>}}%
2021       \bbbl@toggle\bbbl@captionslist
2022     \fi
2023   \fi}
2024 %^^A \def\bbbl@setcaption@s#1#2#3{} % 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.

```

2025 \bbbl@trace{Macros related to glyphs}
2026 \def\set@low@box#1{\setbox\tw@ \hbox{,}\setbox\z@ \hbox{#1}%
2027   \dimen\z@ \ht\z@ \advance\dimen\z@ -\ht\tw@%
2028   \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.

```

2029 \def\save@s@f@q#1{\leavevmode
2030   \begingroup
2031   \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2032   \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.

```
2033 \ProvideTextCommand{\quotedblbase}{OT1}{%
2034 \save@sf@q{\set@low@box{\textquotedblright\}}%
2035 \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.

```
2036 \ProvideTextCommandDefault{\quotedblbase}{%
2037 \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2038 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2039 \save@sf@q{\set@low@box{\textquoteright\}}%
2040 \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.

```
2041 \ProvideTextCommandDefault{\quotesinglbase}{%
2042 \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2043 \ProvideTextCommand{\guillemetleft}{OT1}{%
2044 \ifmmode
2045 \ll
2046 \else
2047 \save@sf@q{\nobreak
2048 \raise.2ex\hbox{\scriptscriptstyle\ll}}\bbl@allowhyphens}%
2049 \fi}
2050 \ProvideTextCommand{\guillemetright}{OT1}{%
2051 \ifmmode
2052 \gg
2053 \else
2054 \save@sf@q{\nobreak
2055 \raise.2ex\hbox{\scriptscriptstyle\gg}}\bbl@allowhyphens}%
2056 \fi}
2057 \ProvideTextCommand{\guillemotleft}{OT1}{%
2058 \ifmmode
2059 \ll
2060 \else
2061 \save@sf@q{\nobreak
2062 \raise.2ex\hbox{\scriptscriptstyle\ll}}\bbl@allowhyphens}%
2063 \fi}
2064 \ProvideTextCommand{\guillemotright}{OT1}{%
2065 \ifmmode
2066 \gg
2067 \else
2068 \save@sf@q{\nobreak
2069 \raise.2ex\hbox{\scriptscriptstyle\gg}}\bbl@allowhyphens}%
2070 \fi}
```

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

```
2071 \ProvideTextCommandDefault{\guillemetleft}{%
2072 \UseTextSymbol{OT1}{\guillemetleft}}
2073 \ProvideTextCommandDefault{\guillemetright}{%
2074 \UseTextSymbol{OT1}{\guillemetright}}
```

```

2075 \ProvideTextCommandDefault{\guillemotleft}{%
2076 \UseTextSymbol{OT1}{\guillemotleft}}
2077 \ProvideTextCommandDefault{\guillemotright}{%
2078 \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2079 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2080 \ifmmode
2081 <%
2082 \else
2083 \save@sf@q{\nobreak
2084 \raise.2ex\hbox{\scriptscriptstyle<$}\bbl@allowhyphens}%
2085 \fi}
2086 \ProvideTextCommand{\guilsinglright}{OT1}{%
2087 \ifmmode
2088 >%
2089 \else
2090 \save@sf@q{\nobreak
2091 \raise.2ex\hbox{\scriptscriptstyle>$}\bbl@allowhyphens}%
2092 \fi}

```

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

```

2093 \ProvideTextCommandDefault{\guilsinglleft}{%
2094 \UseTextSymbol{OT1}{\guilsinglleft}}
2095 \ProvideTextCommandDefault{\guilsinglright}{%
2096 \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.

```

2097 \DeclareTextCommand{\ij}{OT1}{%
2098 i\kern-0.02em\bbl@allowhyphens j}
2099 \DeclareTextCommand{\IJ}{OT1}{%
2100 I\kern-0.02em\bbl@allowhyphens J}
2101 \DeclareTextCommand{\ij}{T1}{\char188}
2102 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2103 \ProvideTextCommandDefault{\ij}{%
2104 \UseTextSymbol{OT1}{\ij}}
2105 \ProvideTextCommandDefault{\IJ}{%
2106 \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).

```

2107 \def\crrtic@{\hrule height0.1ex width0.3em}
2108 \def\crttic@{\hrule height0.1ex width0.33em}
2109 \def\ddj@{%
2110 \setbox0\hbox{d}\dimen@=\ht0
2111 \advance\dimen@lex
2112 \dimen@.45\dimen@
2113 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2114 \advance\dimen@ii.5ex
2115 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2116 \def\DDJ@{%
2117 \setbox0\hbox{D}\dimen@=.55\ht0
2118 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2119 \advance\dimen@ii.15ex % correction for the dash position
2120 \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2121 \dimen@thr@\expandafter\rem@pt\the\fontdimen7\font\dimen@

```

```

2122 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2123 %
2124 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2125 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2126 \ProvideTextCommandDefault{\dj}{%
2127   \UseTextSymbol{OT1}{\dj}}
2128 \ProvideTextCommandDefault{\DJ}{%
2129   \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.

```

2130 \DeclareTextCommand{\SS}{OT1}{SS}
2131 \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
2132 \ProvideTextCommandDefault{\glq}{%
2133   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

```

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

```

2134 \ProvideTextCommand{\grq}{T1}{%
2135   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2136 \ProvideTextCommand{\grq}{TU}{%
2137   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2138 \ProvideTextCommand{\grq}{OT1}{%
2139   \save@sf@q{\kern-.0125em
2140     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2141     \kern.07em\relax}}
2142 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}

```

`\glqq` The ‘german’ double quotes.

```

\grqq
2143 \ProvideTextCommandDefault{\glqq}{%
2144   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

2145 \ProvideTextCommand{\grqq}{T1}{%
2146   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2147 \ProvideTextCommand{\grqq}{TU}{%
2148   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2149 \ProvideTextCommand{\grqq}{OT1}{%
2150   \save@sf@q{\kern-.07em
2151     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2152     \kern.07em\relax}}
2153 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}

```

`\flq` The ‘french’ single guillemets.

```

\frq
2154 \ProvideTextCommandDefault{\flq}{%
2155   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2156 \ProvideTextCommandDefault{\frq}{%
2157   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

`\flqq` The ‘french’ double guillemets.

```

\frqq
2158 \ProvideTextCommandDefault{\flqq}{%
2159   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2160 \ProvideTextCommandDefault{\frqq}{%
2161   \textormath{\guillemetright}{\mbox{\guillemetright}}}

```

4.12.4 Umlauts and tremas

The command `\` needs to have a different effect for different languages. For German for instance, the ‘umlaut’ should be positioned lower than the default position for placing it over the letters a, o, u, A, O and U. When placed over an e, i, E or I it can retain its normal position. For Dutch the same glyph is always placed in the lower position.

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

```
2162 \def\uumlauthigh{%
2163   \def\bbl@umlauta##1{\leavevmode\bgroup%
2164     \accent\csname\fontencoding dqpos\endcsname
2165     ##1\bbl@allowhyphens\egroup}%
2166   \let\bbl@umlaute\bbl@umlauta}
2167 \def\uumlautlow{%
2168   \def\bbl@umlauta{\protect\lower@umlaut}}
2169 \def\uumlautelower{%
2170   \def\bbl@umlaute{\protect\lower@umlaut}}
2171 \umlauthigh
```

`\lower@umlaut` The command `\lower@umlaut` is used to position the `\` closer to the letter. We want the umlaut character lowered, nearer to the letter. To do this we need an extra (*dimen*) register.

```
2172 \expandafter\ifx\csname U@D\endcsname\relax
2173   \csname newdimen\endcsname\U@D
2174 \fi
```

The following code fools TeX’s `make_accent` procedure about the current x-height of the font to force another placement of the umlaut character. First we have to save the current x-height of the font, because we’ll change this font dimension and this is always done globally.

Then we compute the new x-height in such a way that the umlaut character is lowered to the base character. The value of `.45ex` depends on the METAFONT parameters with which the fonts were built. (Just try out, which value will look best.) If the new x-height is too low, it is not changed. Finally we call the `\accent` primitive, reset the old x-height and insert the base character in the argument.

```
2175 \def\lower@umlaut#1{%
2176   \leavevmode\bgroup
2177   \U@D lex%
2178   {\setbox\z@\hbox{%
2179     \char\csname\fontencoding dqpos\endcsname}%
2180     \dimen@ -.45ex\advance\dimen@\ht\z@
2181     \ifdim lex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2182   \accent\csname\fontencoding dqpos\endcsname
2183   \fontdimen5\font\U@D #1%
2184   \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).

```
2185 \AtBeginDocument{%
2186   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2187   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2188   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2189   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{i}}%
2190   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
2191   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
2192   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
2193   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2194   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2195   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
2196   \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.

```
2197 \ifx\l@english\@undefined
2198   \chardef\l@english\z@
2199 \fi
2200 % The following is used to cancel rules in ini files (see Amharic).
2201 \ifx\l@unhyphenated\@undefined
2202   \newlanguage\l@unhyphenated
2203 \fi
```

4.13 Layout

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

```
2204 \bbl@trace{Bidi layout}
2205 \providecommand\IfBabelLayout[3]{#3}%
2206 </package | core>
2207 <*package>
2208 \newcommand\BabelPatchSection[1]{%
2209   \@ifundefined{#1}{}{%
2210     \bbl@exp{\let<bbl@ss@#1>\<#1>}%
2211     \@namedef{#1}{%
2212       \ifstar{\bbl@presec@#1}%
2213         {\@dblarg{\bbl@presec@#1}}}}%
2214 \def\bbl@presec@#1[#2]#3{%
2215   \bbl@exp{%
2216     \\select@language@x{\bbl@main@language}%
2217     \\bbl@cs{sspre@#1}%
2218     \\bbl@cs{ss@#1}%
2219     [\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
2220     {\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
2221     \\select@language@x{\languagename}}%
2222 \def\bbl@presec@s#1#2{%
2223   \bbl@exp{%
2224     \\select@language@x{\bbl@main@language}%
2225     \\bbl@cs{sspre@#1}%
2226     \\bbl@cs{ss@#1}*%
2227     {\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
2228     \\select@language@x{\languagename}}%
2229 \IfBabelLayout{sectioning}%
2230   {\BabelPatchSection{part}%
2231     \BabelPatchSection{chapter}%
2232     \BabelPatchSection{section}%
2233     \BabelPatchSection{subsection}%
2234     \BabelPatchSection{subsubsection}%
2235     \BabelPatchSection{paragraph}%
2236     \BabelPatchSection{subparagraph}%
2237     \def\babel@toc#1{%
2238       \select@language@x{\bbl@main@language}}}}%
2239 \IfBabelLayout{captions}%
2240   {\BabelPatchSection{caption}}}%
2241 </package>
2242 <*package | 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:

```
2243 \bbl@trace{Input engine specific macros}
2244 \ifcase\bbl@engine
2245   \input txtbabel.def
2246 \or
2247   \input luababel.def
```

```

2248 \or
2249 \input xebabel.def
2250 \fi
2251 \providecommand\babelfont{\bbl@error{only-lua-xe}{}}{}{}
2252 \providecommand\babelprehyphenation{\bbl@error{only-lua}{}}{}{}
2253 \ifx\babelposthyphenation\@undefined
2254 \let\babelposthyphenation\babelprehyphenation
2255 \let\babelpatterns\babelprehyphenation
2256 \let\babelcharproperty\babelprehyphenation
2257 \fi

```

4.15 Creating and modifying languages

Continue with \LaTeX only.

`\babelprovide` is a general purpose tool for creating and modifying languages. It creates the language infrastructure, and loads, if requested, an ini file. It may be used in conjunction to previously loaded ldf files.

```

2258 </package | core>
2259 <*package>
2260 \bbl@trace{Creating languages and reading ini files}
2261 \let\bbl@extend@ini@gobble
2262 \newcommand\babelprovide[2][]{%
2263   \let\bbl@savelangname\language
2264   \edef\bbl@savelocaleid{\the\localeid}%
2265   % Set name and locale id
2266   \edef\language{#2}%
2267   \bbl@id@assign
2268   % Initialize keys
2269   \bbl@vforeach{captions,date,import,main,script,language,%
2270     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2271     mapdigits,intraspace,intrapenalty,onchar,transforms,alph,%
2272     Alph,labels,labels*,calendar,date,casing,interchar}%
2273     {\bbl@csarg\let{KVP@##1}\@nnil}%
2274   \global\let\bbl@release@transforms\@empty
2275   \global\let\bbl@release@casing\@empty
2276   \let\bbl@calendars\@empty
2277   \global\let\bbl@inidata\@empty
2278   \global\let\bbl@extend@ini@gobble
2279   \global\let\bbl@included@inis\@empty
2280   \gdef\bbl@key@list{;}%
2281   \bbl@forkv{#1}{%
2282     \in@{/}{##1}% With /, (re)sets a value in the ini
2283     \ifin@
2284       \global\let\bbl@extend@ini\bbl@extend@ini@aux
2285       \bbl@renewinikey##1\@{##2}%
2286     \else
2287       \bbl@csarg\ifx{KVP@##1}\@nnil\else
2288         \bbl@error{unknown-provide-key}{##1}{}}{}%
2289     \fi
2290     \bbl@csarg\def{KVP@##1}{##2}%
2291   \fi}%
2292   \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2293   \bbl@ifunset{date#2}\z@{\bbl@ifunset{\bbl@llevel@#2}\@ne\tw@}%
2294   % == init ==
2295   \ifx\bbl@screset\@undefined
2296     \bbl@ldfinit
2297   \fi
2298   % == date (as option) ==
2299   % \ifx\bbl@KVP@date\@nnil\else
2300   % \fi
2301   % ==
2302   \let\bbl@lbfkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2303   \ifcase\bbl@howloaded

```

```

2304 \let\bbkflag\@empty % new
2305 \else
2306 \ifx\bbkflag\@empty
2307 \let\bbkflag\@empty
2308 \fi
2309 \ifx\bbkflag\@empty
2310 \let\bbkflag\@empty
2311 \fi
2312 \fi
2313 % == import, captions ==
2314 \ifx\bbkflag\@empty
2315 \bbk@exp{\bbk@ifblank{\bbk@KVP@import}}%
2316 {\ifx\bbk@initload\relax
2317 \begingroup
2318 \def\BabelBeforeIni##1##2{\gdef\bbk@KVP@import{##1}\endinput}%
2319 \bbk@input@texini{##2}%
2320 \endgroup
2321 \else
2322 \xdef\bbk@KVP@import{\bbk@initload}%
2323 \fi}%
2324 {}%
2325 \let\bbk@KVP@date\@empty
2326 \fi
2327 \let\bbk@KVP@captions@\bbk@KVP@captions %^^A A dirty hack
2328 \ifx\bbk@KVP@captions\@empty
2329 \let\bbk@KVP@captions\bbk@KVP@import
2330 \fi
2331 % ==
2332 \ifx\bbk@KVP@transforms\@empty
2333 \bbk@replace\bbk@KVP@transforms{ }{,}%
2334 \fi
2335 % == Load ini ==
2336 \ifcase\bbk@howloaded
2337 \bbk@provide@new{##2}%
2338 \else
2339 \bbk@ifblank{##1}%
2340 {}% With \bbk@load@basic below
2341 {\bbk@provide@renew{##2}}%
2342 \fi
2343 % == include == TODO
2344 % \ifx\bbk@included@inis\@empty\else
2345 % \bbk@replace\bbk@included@inis{ }{,}%
2346 % \bbk@foreach\bbk@included@inis{%
2347 % \openin\bbk@readstream=babel-##1.ini
2348 % \bbk@extend@ini{##2}}%
2349 % \closein\bbk@readstream
2350 % \fi
2351 % Post tasks
2352 % -----
2353 % == subsequent calls after the first provide for a locale ==
2354 \ifx\bbk@inidata\@empty\else
2355 \bbk@extend@ini{##2}%
2356 \fi
2357 % == ensure captions ==
2358 \ifx\bbk@KVP@captions\@empty\else
2359 \bbk@ifunset{\bbk@extracaps@##2}%
2360 {\bbk@exp{\bbk@babelensure[exclude=\bbk@today]{##2}}}%
2361 {\bbk@exp{\bbk@babelensure[exclude=\bbk@today,
2362 include=\bbk@extracaps@##2]{##2}}}%
2363 \bbk@ifunset{\bbk@ensure@language}%
2364 {\bbk@exp{%
2365 \bbk@DeclareRobustCommand\bbk@ensure@language[1]{%
2366 \bbk@foreignlanguage{language}%

```



```

2367     {###1}}}%
2368   {}%
2369   \bbl@exp{%
2370     \\bbl@tglobal\<bbl@ensure@\languagename>%
2371     \\bbl@tglobal\<bbl@ensure@\languagename\space>%
2372   \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.

```

2373   \bbl@load@basic{#2}%
2374   % == script, language ==
2375   % Override the values from ini or defines them
2376   \ifx\bbl@KVP@script@nnil\else
2377     \bbl@csarg\edef{sname#2}{\bbl@KVP@script}%
2378   \fi
2379   \ifx\bbl@KVP@language@nnil\else
2380     \bbl@csarg\edef{lname#2}{\bbl@KVP@language}%
2381   \fi
2382   \ifcase\bbl@engine\or
2383     \bbl@ifunset{bbl@chrng@\languagename}{}%
2384     {\directlua{
2385       Babel.set_chranges_b('\bbl@cl{sbcpr}', '\bbl@cl{chrng}') }}%
2386   \fi
2387   % == onchar ==
2388   \ifx\bbl@KVP@onchar@nnil\else
2389     \bbl@luahyphenate
2390     \bbl@exp{%
2391       \\AddToHook{env/document/before}{\select@language{#2}}}%
2392     \directlua{
2393       if Babel.locale_mapped == nil then
2394         Babel.locale_mapped = true
2395         Babel.linebreaking.add_before(Babel.locale_map, 1)
2396         Babel.loc_to_scr = {}
2397         Babel.chr_to_loc = Babel.chr_to_loc or {}
2398       end
2399       Babel.locale_props[\the\localeid].letters = false
2400     }%
2401     \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
2402   \ifin@
2403     \directlua{
2404       Babel.locale_props[\the\localeid].letters = true
2405     }%
2406   \fi
2407   \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2408   \ifin@
2409     \ifx\bbl@starthyphens@undefined % Needed if no explicit selection
2410       \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
2411     \fi
2412     \bbl@exp{\\bbl@add\\bbl@starthyphens
2413       {\bbl@patterns@lua{\languagename}}}%
2414     %^^A add error/warning if no script
2415     \directlua{
2416       if Babel.script_blocks['\bbl@cl{sbcpr}'] then
2417         Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbl@cl{sbcpr}']
2418         Babel.locale_props[\the\localeid].lg = \the@nameuse{l@\languagename}\space
2419       end
2420     }%
2421   \fi
2422   \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2423   \ifin@
2424     \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{%
2425     \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{%

```

```

2426 \directlua{
2427   if Babel.script_blocks['\bbl@cl{sbc}'] then
2428     Babel.loc_to_scr[\the\localeid] =
2429     Babel.script_blocks['\bbl@cl{sbc}']
2430   end}%
2431 \ifx\bbl@mapselect\undefined % TODO. almost the same as mapfont
2432 \AtBeginDocument{%
2433   \bbl@patchfont{\bbl@mapselect}}%
2434   {\selectfont}}%
2435 \def\bbl@mapselect{%
2436   \let\bbl@mapselect\relax
2437   \edef\bbl@prefontid{\fontid\font}}%
2438 \def\bbl@mapdir##1{%
2439   \begingroup
2440     \setbox\z@\hbox{% Force text mode
2441       \def\language{##1}}%
2442     \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2443     \bbl@switchfont
2444     \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2445       \directlua{
2446         Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
2447         [\bbl@prefontid] = \fontid\font\space}%
2448       \fi}%
2449   \endgroup}%
2450 \fi
2451 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir\language}}%
2452 \fi
2453 % TODO - catch non-valid values
2454 \fi
2455 % == mapfont ==
2456 % For bidi texts, to switch the font based on direction
2457 \ifx\bbl@KVP@mapfont\@nnil\else
2458   \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}%
2459   {\bbl@error{unknown-mapfont}}{}%
2460   \bbl@ifunset{\bbl@lsys\language}{\bbl@provide@lsys\language}}%
2461   \bbl@ifunset{\bbl@wdir\language}{\bbl@provide@dirs\language}}%
2462 \ifx\bbl@mapselect\undefined % TODO. See onchar.
2463 \AtBeginDocument{%
2464   \bbl@patchfont{\bbl@mapselect}}%
2465   {\selectfont}}%
2466 \def\bbl@mapselect{%
2467   \let\bbl@mapselect\relax
2468   \edef\bbl@prefontid{\fontid\font}}%
2469 \def\bbl@mapdir##1{%
2470   {\def\language{##1}}%
2471   \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2472   \bbl@switchfont
2473   \directlua{Babel.fontmap
2474     [\the\csname bbl@wdir@##1\endcsname]%
2475     [\bbl@prefontid]=\fontid\font}}%
2476 \fi
2477 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir\language}}%
2478 \fi
2479 % == Line breaking: intraspace, intrapenalty ==
2480 % For CJK, East Asian, Southeast Asian, if interspace in ini
2481 \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2482   \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2483 \fi
2484 \bbl@provide@intraspace
2485 % == Line breaking: CJK quotes == %^A -> @extras
2486 \ifcase\bbl@engine\or
2487   \bbl@xin{/c}{\bbl@cl{lnbrk}}%
2488 \ifin@

```

```

2489 \bbl@ifunset{bbl@quote@\languagename}{}%
2490 {\directlua{
2491   Babel.locale_props[\the\localeid].cjk_quotes = {}
2492   local cs = 'op'
2493   for c in string.utfvalues(
2494     [[\csname bbl@quote@\languagename\endcsname]]) do
2495     if Babel.cjk_characters[c].c == 'qu' then
2496       Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2497     end
2498     cs = ( cs == 'op' ) and 'cl' or 'op'
2499   end
2500 }}%
2501 \fi
2502 \fi
2503 % == Line breaking: justification ==
2504 \ifx\bbl@KVP@justification\@nnil\else
2505   \let\bbl@KVP@linebreaking\bbl@KVP@justification
2506 \fi
2507 \ifx\bbl@KVP@linebreaking\@nnil\else
2508   \bbl@xin@{,\bbl@KVP@linebreaking,}%
2509   {,elongated,kashida,cjk,padding,unhyphenated,}%
2510   \ifin@
2511     \bbl@csarg\xdef
2512     {\lbrk@\languagename}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2513   \fi
2514 \fi
2515 \bbl@xin@{/e}{/\bbl@cl{\lbrk}}%
2516 \ifin@else\bbl@xin@{/k}{/\bbl@cl{\lbrk}}\fi
2517 \ifin@\bbl@arabicjust\fi
2518 \bbl@xin@{/p}{/\bbl@cl{\lbrk}}%
2519 \ifin@\AtBeginDocument{\@nameuse{bbl@tibetanjust}}\fi
2520 % == Line breaking: hyphenate.other.(locale|script) ==
2521 \ifx\bbl@lbrkflag\@empty
2522   \bbl@ifunset{bbl@hyotl@\languagename}{}%
2523   {\bbl@csarg\bbl@replace{hyotl@\languagename}{ }{,}}%
2524   \bbl@startcommands*\languagename}{}%
2525   \bbl@csarg\bbl@foreach{hyotl@\languagename}{%
2526     \ifcase\bbl@engine
2527       \ifnum##1<257
2528         \SetHyphenMap{\BabelLower{##1}{##1}}%
2529       \fi
2530     \else
2531       \SetHyphenMap{\BabelLower{##1}{##1}}%
2532     \fi}%
2533   \bbl@endcommands}%
2534 \bbl@ifunset{bbl@hyots@\languagename}{}%
2535   {\bbl@csarg\bbl@replace{hyots@\languagename}{ }{,}}%
2536   \bbl@csarg\bbl@foreach{hyots@\languagename}{%
2537     \ifcase\bbl@engine
2538       \ifnum##1<257
2539         \global\lccode##1=##1\relax
2540       \fi
2541     \else
2542       \global\lccode##1=##1\relax
2543     \fi}}%
2544 \fi
2545 % == Counters: maparabic ==
2546 % Native digits, if provided in ini (TeX level, xe and lua)
2547 \ifcase\bbl@engine\else
2548   \bbl@ifunset{bbl@dgnat@\languagename}{}%
2549   {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
2550     \expandafter\expandafter\expandafter
2551     \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname

```

```

2552     \ifx\bbbl@KVP@maparabic\@nnil\else
2553     \ifx\bbbl@latinarabic\@undefined
2554     \expandafter\let\expandafter\@arabic
2555     \csname bbl@counter@languagename\endcsname
2556     \else % ie, if layout=counters, which redefines \@arabic
2557     \expandafter\let\expandafter\bbbl@latinarabic
2558     \csname bbl@counter@languagename\endcsname
2559     \fi
2560     \fi
2561     \fi}%
2562 \fi
2563 % == Counters: mapdigits ==
2564 % > luababel.def
2565 % == Counters: alph, Alph ==
2566 \ifx\bbbl@KVP@alph\@nnil\else
2567     \bbl@exp{%
2568         \\bbbl@add\<bbl@preextras@\languagename>{%
2569             \\babel@save\\ \@alph
2570             \let\\ \@alph\<bbl@cntr@\bbbl@KVP@alph @\languagename>}}%
2571 \fi
2572 \ifx\bbbl@KVP@Alph\@nnil\else
2573     \bbl@exp{%
2574         \\bbbl@add\<bbl@preextras@\languagename>{%
2575             \\babel@save\\ \@Alph
2576             \let\\ \@Alph\<bbl@cntr@\bbbl@KVP@Alph @\languagename>}}%
2577 \fi
2578 % == Casing ==
2579 \bbl@release@casing
2580 \ifx\bbbl@KVP@casing\@nnil\else
2581     \bbl@csarg\xdef{casing@\languagename}%
2582     {\@nameuse{bbl@casing@\languagename}\bbl@maybextx\bbl@KVP@casing}%
2583 \fi
2584 % == Calendars ==
2585 \ifx\bbbl@KVP@calendar\@nnil
2586     \edef\bbl@KVP@calendar{\bbl@cl{calpr}}}%
2587 \fi
2588 \def\bbl@tempe##1 ##2\@{% Get first calendar
2589     \def\bbl@tempa{##1}}%
2590     \bbl@exp{\\bbbl@tempe\bbl@KVP@calendar\space\\ \@}%
2591 \def\bbl@tempe##1.##2.##3\@{%
2592     \def\bbl@tempc{##1}%
2593     \def\bbl@tempb{##2}}%
2594 \expandafter\bbl@tempe\bbl@tempa.\@
2595 \bbl@csarg\edef{calpr@\languagename}{%
2596     \ifx\bbl@tempc\@empty\else
2597     calendar=\bbl@tempc
2598     \fi
2599     \ifx\bbl@tempb\@empty\else
2600     ,variant=\bbl@tempb
2601     \fi}%
2602 % == engine specific extensions ==
2603 % Defined in XXXbabel.def
2604 \bbl@provide@extra{#2}%
2605 % == require.babel in ini ==
2606 % To load or reload the babel-*.tex, if require.babel in ini
2607 \ifx\bbbl@beforestart\relax\else % But not in doc aux or body
2608     \bbl@ifunset{bbl@rqtex@\languagename}{}%
2609     {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
2610         \let\BabelBeforeIni\@gobbletwo
2611         \chardef\atcatcode=\catcode\@
2612         \catcode\@=11\relax
2613         \def\CurrentOption{#2}%
2614         \bbl@input@texini{\bbl@cs{rqtex@\languagename}}}%

```

```

2615     \catcode`\@=\atcatcode
2616     \let\atcatcode\relax
2617     \global\bbbl@csarg\let{rqtex@\language}\relax
2618     \fi}%
2619     \bbbl@foreach\bbbl@calendars{%
2620     \bbbl@ifunset{bbbl@ca##1}{%
2621     \chardef\atcatcode=\catcode`\@
2622     \catcode`\@=11\relax
2623     \InputIfFileExists{babel-ca-##1.tex}{}{}%
2624     \catcode`\@=\atcatcode
2625     \let\atcatcode\relax}%
2626     {}}%
2627 \fi
2628 % == frenchspacing ==
2629 \ifcase\bbbl@howloaded\in@true\else\in@false\fi
2630 \ifin@else\bbbl@xin@{typography/frenchspacing}{\bbbl@key@list}\fi
2631 \ifin@
2632     \bbbl@extras@wrap{\bbbl@pre@fs}%
2633     {\bbbl@pre@fs}%
2634     {\bbbl@post@fs}%
2635 \fi
2636 % == transforms ==
2637 % > luabel.def
2638 \def\CurrentOption{#2}%
2639 \@nameuse{bbbl@icsave@#2}%
2640 % == main ==
2641 \ifx\bbbl@KVP@main@\nnil % Restore only if not 'main'
2642     \let\language\bbbl@savelangname
2643     \chardef\localeid\bbbl@savelocaleid\relax
2644 \fi
2645 % == hyphenrules (apply if current) ==
2646 \ifx\bbbl@KVP@hyphenrules@\nnil\else
2647     \ifnum\bbbl@savelocaleid=\localeid
2648     \language\@nameuse{l@\language}%
2649 \fi
2650 \fi}

```

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

```

2651 \def\bbbl@provide@new#1{%
2652     \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2653     \@namedef{extras#1}{}%
2654     \@namedef{noextras#1}{}%
2655     \bbbl@startcommands*{#1}{captions}%
2656     \ifx\bbbl@KVP@captions@\nnil % and also if import, implicit
2657     \def\bbbl@tempb##1{% elt for \bbbl@captionslist
2658     \ifx##1@\nnil\else
2659     \bbbl@exp{%
2660     \\SetString\\##1{%
2661     \\bbbl@nocaption{\bbbl@stripslash##1}{#1\bbbl@stripslash##1}}%
2662     \expandafter\bbbl@tempb
2663     \fi}%
2664     \expandafter\bbbl@tempb\bbbl@captionslist@\nnil
2665 \else
2666     \ifx\bbbl@initoload\relax
2667     \bbbl@read@ini{\bbbl@KVP@captions}2% % Here letters cat = 11
2668     \else
2669     \bbbl@read@ini{\bbbl@initoload}2% % Same
2670     \fi
2671 \fi
2672 \StartBabelCommands*{#1}{date}%
2673 \ifx\bbbl@KVP@date@\nnil
2674     \bbbl@exp{%

```

```

2675     \\SetString\\today{\\bb@nocaption{today}{#1today}}}%
2676   \\else
2677     \\bb@savetoday
2678     \\bb@savedate
2679   \\fi
2680 \\bb@endcommands
2681 \\bb@load@basic{#1}%
2682 % == hyphenmins == (only if new)
2683 \\bb@exp{%
2684   \\gdef<#1hyphenmins>{%
2685     {\\bb@ifunset{bb@lfthm@#1}{2}{\\bb@cs{lfthm@#1}}}%
2686     {\\bb@ifunset{bb@rgthm@#1}{3}{\\bb@cs{rgthm@#1}}}}}%
2687 % == hyphenrules (also in renew) ==
2688 \\bb@provide@hyphens{#1}%
2689 \\ifx\\bb@KVP@main@\\nnil\\else
2690   \\expandafter\\main@language\\expandafter{#1}%
2691 \\fi}
2692 %
2693 \\def\\bb@provide@renew#1{%
2694   \\ifx\\bb@KVP@captions@\\nnil\\else
2695     \\StartBabelCommands*{#1}{captions}%
2696     \\bb@read@ini{\\bb@KVP@captions}2%   % Here all letters cat = 11
2697     \\EndBabelCommands
2698   \\fi
2699   \\ifx\\bb@KVP@date@\\nnil\\else
2700     \\StartBabelCommands*{#1}{date}%
2701     \\bb@savetoday
2702     \\bb@savedate
2703     \\EndBabelCommands
2704   \\fi
2705 % == hyphenrules (also in new) ==
2706 \\ifx\\bb@lbkflag@\\empty
2707   \\bb@provide@hyphens{#1}%
2708 \\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.

```

2709 \\def\\bb@load@basic#1{%
2710   \\ifcase\\bb@howloaded\\or\\or
2711     \\ifcase\\csname bbl@llevel@\\languagename\\endcsname
2712       \\bb@csarg\\let{lname@\\languagename}\\relax
2713     \\fi
2714   \\fi
2715   \\bb@ifunset{bb@lname@#1}%
2716   {\\def\\BabelBeforeIni##1##2{%
2717     \\begingroup
2718       \\let\\bb@ini@captions@aux\\gobbletwo
2719       \\def\\bb@inidate ####1.####2.####3.####4\\relax ####5####6}%
2720     \\bb@read@ini{##1}1%
2721     \\ifx\\bb@initoload\\relax\\endinput\\fi
2722     \\endgroup}%
2723     \\begingroup      % boxed, to avoid extra spaces:
2724     \\ifx\\bb@initoload\\relax
2725       \\bb@input@texini{##1}%
2726     \\else
2727       \\setbox\\z@\\hbox{\\BabelBeforeIni{\\bb@initoload}}}%
2728     \\fi
2729     \\endgroup}%
2730   {}}

```

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.

```

2731 \\def\\bb@provide@hyphens#1{%

```

```

2732 \@tempcnta\m@ne % a flag
2733 \ifx\bbk@KVP@hyphenrules\@nnil\else
2734 \bbk@replace\bbk@KVP@hyphenrules{ }{,}%
2735 \bbk@foreach\bbk@KVP@hyphenrules{%
2736 \ifnum\@tempcnta=\m@ne % if not yet found
2737 \bbk@ifsamestring{##1}{+}%
2738 {\bbk@carg\addlanguage{l@##1}}%
2739 }%
2740 \bbk@ifunset{l@##1}% After a possible +
2741 {}%
2742 {\@tempcnta\@nameuse{l@##1}}%
2743 \fi}%
2744 \ifnum\@tempcnta=\m@ne
2745 \bbk@warning{%
2746 Requested 'hyphenrules' for '\language' not found:\%
2747 \bbk@KVP@hyphenrules.\%
2748 Using the default value. Reported}%
2749 \fi
2750 \fi
2751 \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found
2752 \ifx\bbk@KVP@captions@\@nnil % TODO. Hackish. See above.
2753 \bbk@ifunset{\bbk@hyphr@#1}{}% use value in ini, if exists
2754 {\bbk@exp{\bbk@ifblank{\bbk@cs{hyphr@#1}}}%
2755 }%
2756 {\bbk@ifunset{l@\bbk@cl{hyphr}}}%
2757 {}% if hyphenrules found:
2758 {\@tempcnta\@nameuse{l@\bbk@cl{hyphr}}}%
2759 \fi
2760 \fi
2761 \bbk@ifunset{l@#1}%
2762 {\ifnum\@tempcnta=\m@ne
2763 \bbk@carg\adddialect{l@#1}\language
2764 \else
2765 \bbk@carg\adddialect{l@#1}\@tempcnta
2766 \fi}%
2767 {\ifnum\@tempcnta=\m@ne\else
2768 \global\bbk@carg\chardef{l@#1}\@tempcnta
2769 \fi}}

```

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

```

2770 \def\bbk@input@texini#1{%
2771 \bbk@bsphack
2772 \bbk@exp{%
2773 \catcode`\\%=14 \catcode`\\%=0
2774 \catcode`\\#{1 \catcode`\\}=2
2775 \lowercase{\\InputIfFileExists{babel-#1.tex}{}}}%
2776 \catcode`\\%=\the\catcode`\% \relax
2777 \catcode`\\=\the\catcode`\% \relax
2778 \catcode`\\{=\the\catcode`\% \relax
2779 \catcode`\\}= \the\catcode`\% \relax}%
2780 \bbk@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 \bbk@read@ini.

```

2781 \def\bbk@inline#1\bbk@inline{%
2782 \@ifnextchar[\bbk@inisect{\@ifnextchar;\bbk@iniskip\bbk@inistore}#1\@{}% ]
2783 \def\bbk@inisect[#1]#2\@{\def\bbk@section{#1}}
2784 \def\bbk@iniskip#1\@{\% if starts with ;
2785 \def\bbk@inistore#1=#2\@{\% full (default)
2786 \bbk@trim@def\bbk@tempa{#1}%
2787 \bbk@trim\toks@{#2}%
2788 \bbk@xin@{\bbk@section/\bbk@tempa;}{\bbk@key@list}%
2789 \ifin@else

```

```

2790 \bbl@xin@{,identification/include.}%
2791     {,\bbl@section/\bbl@tempa}%
2792 \ifin@\xdef\bbl@included@inis{\the\toks@}\fi
2793 \bbl@exp{%
2794     \\g@addto@macro\\bbl@inidata{%
2795         \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2796 \fi}
2797 \def\bbl@inistore@min#1=#2\@@{% minimal (maybe set in \bbl@read@ini)
2798 \bbl@trim@def\bbl@tempa{#1}%
2799 \bbl@trim\toks@{#2}%
2800 \bbl@xin@{.identification.}{.\bbl@section.}%
2801 \ifin@
2802     \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2803         \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2804 \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.

```

2805 \def\bbl@loop@ini{%
2806 \loop
2807     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2808     \endlinechar\m@ne
2809     \read\bbl@readstream to \bbl@line
2810     \endlinechar\^^M
2811     \ifx\bbl@line\@empty\else
2812         \expandafter\bbl@iniline\bbl@line\bbl@iniline
2813     \fi
2814 \repeat}
2815 \ifx\bbl@readstream\@undefined
2816 \csname newread\endcsname\bbl@readstream
2817 \fi
2818 \def\bbl@read@ini#1#2{%
2819 \global\let\bbl@extend@ini\@gobble
2820 \openin\bbl@readstream=babel-#1.ini
2821 \ifeof\bbl@readstream
2822     \bbl@error{no-ini-file}{#1}{}%
2823 \else
2824     % == Store ini data in \bbl@inidata ==
2825     \catcode\|=12 \catcode\|=12 \catcode\|=12 \catcode\&=12
2826     \catcode\;=12 \catcode\|=12 \catcode\%=14 \catcode\-=12
2827     \bbl@info{Importing
2828         \ifcase#2font and identification \or basic \fi
2829         data for \languagename}%
2830     from babel-#1.ini. Reported}%
2831 \ifnum#2=\z@
2832     \global\let\bbl@inidata\@empty
2833     \let\bbl@inistore\bbl@inistore@min % Remember it's local
2834 \fi
2835 \def\bbl@section{identification}%
2836 \bbl@exp{\\bbl@inistore tag.ini=#1\\@@}%
2837 \bbl@inistore load.level=#2\@@
2838 \bbl@loop@ini
2839 % == Process stored data ==
2840 \bbl@csarg\xdef{lini@\languagename}{#1}%
2841 \bbl@read@ini@aux
2842 % == 'Export' data ==
2843 \bbl@ini@exports{#2}%
2844 \global\bbl@csarg\let{inidata@\languagename}\bbl@inidata
2845 \global\let\bbl@inidata\@empty

```



```

2846 \bbl@exp{\bbl@add@list\bbl@ini@loaded{\language}}%
2847 \bbl@tglobal\bbl@ini@loaded
2848 \fi
2849 \closein\bbl@readstream}
2850 \def\bbl@read@ini@aux{%
2851 \let\bbl@savestrings\empty
2852 \let\bbl@savetoday\empty
2853 \let\bbl@savedate\empty
2854 \def\bbl@elt##1##2##3{%
2855 \def\bbl@section{##1}%
2856 \in@{=date.}{=##1}% Find a better place
2857 \ifin@
2858 \bbl@ifunset{bbl@inikv@##1}%
2859 {\bbl@ini@calendar{##1}}%
2860 {}%
2861 \fi
2862 \bbl@ifunset{bbl@inikv@##1}{}%
2863 {\csname bbl@inikv@##1\endcsname{##2}{##3}}%
2864 \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.

```

2865 \def\bbl@extend@ini@aux#1{%
2866 \bbl@startcommands*{#1}{captions}%
2867 % Activate captions/... and modify exports
2868 \bbl@csarg\def{inikv@captions.licr}##1##2{%
2869 \setlocalecaption{#1}{##1}{##2}}%
2870 \def\bbl@inikv@captions##1##2{%
2871 \bbl@ini@captions@aux{##1}{##2}}%
2872 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2873 \def\bbl@exportkey##1##2##3{%
2874 \bbl@ifunset{bbl@kv@##2}{}%
2875 {\expandafter\ifx\csname bbl@kv@##2\endcsname\empty\else
2876 \bbl@exp{\global\let<bbl@##1@\language>\<bbl@kv@##2>}}%
2877 \fi}}%
2878 % As with \bbl@read@ini, but with some changes
2879 \bbl@read@ini@aux
2880 \bbl@ini@exports\tw@
2881 % Update inidata@lang by pretending the ini is read.
2882 \def\bbl@elt##1##2##3{%
2883 \def\bbl@section{##1}%
2884 \bbl@iniline##2=##3\bbl@iniline}%
2885 \csname bbl@inidata@#1\endcsname
2886 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2887 \StartBabelCommands*{#1}{date}% And from the import stuff
2888 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2889 \bbl@savetoday
2890 \bbl@savedate
2891 \bbl@endcommands}

```

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

```

2892 \def\bbl@ini@calendar#1{%
2893 \lowercase{\def\bbl@tempa{=#1=}}%
2894 \bbl@replace\bbl@tempa{=date.gregorian}{}%
2895 \bbl@replace\bbl@tempa{=date.}{}%
2896 \in@{.licr}=#1=%
2897 \ifin@
2898 \ifcase\bbl@engine
2899 \bbl@replace\bbl@tempa{.licr}{}%
2900 \else
2901 \let\bbl@tempa\relax
2902 \fi
2903 \fi
2904 \ifx\bbl@tempa\relax\else

```

```

2905 \bbl@replace\bbl@tempa{=}{}%
2906 \ifx\bbl@tempa@empty\else
2907   \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2908 \fi
2909 \bbl@exp{%
2910   \def<bbl@inikv@#1>####1####2{%
2911     \\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2912 \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).

```

2913 \def\bbl@renewinikey#1/#2\@#3{%
2914   \edef\bbl@tempa{\zap@space #1 \@empty}%   section
2915   \edef\bbl@tempb{\zap@space #2 \@empty}%   key
2916   \bbl@trim\toks@{#3}%                       value
2917   \bbl@exp{%
2918     \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2919     \\g@addto@macro\\bbl@inidata{%
2920       \\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.

```

2921 \def\bbl@exportkey#1#2#3{%
2922   \bbl@ifunset{bbl@kv@#2}%
2923     {\bbl@csarg\gdef{#1@\languagename}{#3}}%
2924     {\expandafter\ifx\csname bbl@kv@#2\endcsname\empty
2925       \bbl@csarg\gdef{#1@\languagename}{#3}%
2926       \else
2927         \bbl@exp{\global\let<bbl@#1@\languagename>\<bbl@kv@#2>}}%
2928   \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.

```

2929 \def\bbl@iniwarning#1{%
2930   \bbl@ifunset{bbl@kv@identification.warning#1}{}%
2931   {\bbl@warning{%
2932     From babel-\bbl@cs{lini@\languagename}.ini:\\%
2933     \bbl@cs{kv@identification.warning#1}\\%
2934     Reported }}}
2935 %
2936 \let\bbl@release@transforms@empty
2937 \let\bbl@release@casing@empty
2938 \def\bbl@ini@exports#1{%
2939   % Identification always exported
2940   \bbl@iniwarning{}%
2941   \ifcase\bbl@engine
2942     \bbl@iniwarning{.pdflatex}%
2943   \or
2944     \bbl@iniwarning{.lualatex}%
2945   \or
2946     \bbl@iniwarning{.xelatex}%
2947   \fi%
2948   \bbl@exportkey{llevel}{identification.load.level}{}%
2949   \bbl@exportkey{elname}{identification.name.english}{}%
2950   \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%
2951     {\csname bbl@elname@\languagename\endcsname}}%
2952   \bbl@exportkey{tbcp}{identification.tag.bcp47}{}%
2953   % Somewhat hackish. TODO:

```

```

2954 \bbl@exportkey{casing}{identification.tag.bcp47}{}%
2955 \bbl@exportkey{lbcpl}{identification.language.tag.bcp47}{}%
2956 \bbl@exportkey{lotf}{identification.tag.opentype}{dfLT}%
2957 \bbl@exportkey{esname}{identification.script.name}{}%
2958 \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
2959   {csname bbl@esname@languagename\endcsname}}%
2960 \bbl@exportkey{sbcpl}{identification.script.tag.bcp47}{}%
2961 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2962 \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2963 \bbl@exportkey{vbcpl}{identification.variant.tag.bcp47}{}%
2964 \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%
2965 \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
2966 \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
2967 % Also maps bcp47 -> languagename
2968 \ifbbl@bcptoname
2969   \bbl@csarg\xdef{bcp@map@bbl@cl{tbcpl}}{languagename}%
2970 \fi
2971 \ifcase\bbl@engine\or
2972   \directlua{%
2973     babel.locale_props[\the\bbl@cs{id@languagename}].script
2974     = '\bbl@cl{sbcpl}'}%
2975 \fi
2976 % Conditional
2977 \ifnum#1>\z@ % 0 = only info, 1, 2 = basic, (re)new
2978   \bbl@exportkey{calpr}{date.calendar.preferred}{}%
2979   \bbl@exportkey{lncr}{typography.linebreaking}{h}%
2980   \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2981   \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
2982   \bbl@exportkey{rgtm}{typography.righthyphenmin}{3}%
2983   \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2984   \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2985   \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2986   \bbl@exportkey{intsp}{typography.intraspaces}{u}%
2987   \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2988   \bbl@exportkey{chrng}{characters.ranges}{}%
2989   \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2990   \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2991   \ifnum#1=\tw@ % only (re)new
2992     \bbl@exportkey{rqtex}{identification.require.babel}{}%
2993     \bbl@toggle\bbl@savetoday
2994     \bbl@toggle\bbl@savestate
2995     \bbl@savestrings
2996   \fi
2997 \fi}

```

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

```

2998 \def\bbl@inikv#1#2{%      key=value
2999   \toks@{#2}%             This hides #'s from ini values
3000   \bbl@csarg\edef{@kv@bbl@section.#1}{\the\toks@}}

```

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

```

3001 \let\bbl@inikv@identification\bbl@inikv
3002 \let\bbl@inikv@date\bbl@inikv
3003 \let\bbl@inikv@typography\bbl@inikv
3004 \let\bbl@inikv@numbers\bbl@inikv

```

The characters section also stores the values, but casing is treated in a different fashion. Much like transforms, a set of commands calling the parser are stored in \bbl@release@casing, which is executed in \babelprovide.

```

3005 \def\bbl@maybextx{-\bbl@csarg\ifx{extx@languagename}\empty x-\fi}
3006 \def\bbl@inikv@characters#1#2{%
3007   \bbl@ifsamestring{#1}{casing}% eg, casing = uV
3008   {\bbl@exp{%
3009     \g@addto@macro{\bbl@release@casing{%

```

```

3010     \\bbl@casemapping}{\languagename}{\unexpanded{#2}}}%
3011 {\in@{ $casing. }{ $#1 }% eg, casing.Uv = uV
3012   \ifin@
3013     \lowercase{\def\bbl@tempb{#1}}%
3014     \bbl@replace\bbl@tempb{casing.}{}%
3015     \bbl@exp{\\g@addto@macro\\bbl@release@casing{%
3016       \\bbl@casemapping
3017         {\\bbl@maybextx\bbl@tempb}{\languagename}{\unexpanded{#2}}}%
3018     \else
3019       \bbl@inikv{#1}{#2}%
3020   \fi}}

```

Additive numerals require an additional definition. When .1 is found, two macros are defined – the basic one, without .1 called by \localenumerals, and another one preserving the trailing .1 for the ‘units’.

```

3021 \def\bbl@inikv@counters#1#2{%
3022   \bbl@ifsamestring{#1}{digits}%
3023   {\bbl@error{digits-is-reserved}{}}}%
3024   {%
3025   \def\bbl@tempc{#1}%
3026   \bbl@trim@def{\bbl@tempb*}{#2}%
3027   \in@{.1$}{#1$}%
3028   \ifin@
3029     \bbl@replace\bbl@tempc{.1}{}%
3030     \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\languagename}{%
3031       \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3032   \fi
3033   \in@{.F.}{#1}%
3034   \ifin@\else\in@{.S.}{#1}\fi
3035   \ifin@
3036     \bbl@csarg\protected@xdef{cntr@#1@\languagename}{\bbl@tempb*}%
3037   \else
3038     \toks@{}% Required by \bbl@builddifcase, which returns \bbl@tempa
3039     \expandafter\bbl@builddifcase\bbl@tempb* \ \ % Space after \
3040     \bbl@csarg{\global\expandafter\let}{cntr@#1@\languagename}\bbl@tempa
3041   \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.

```

3042 \ifcase\bbl@engine
3043   \bbl@csarg\def{inikv@captions.licr}#1#2{%
3044     \bbl@ini@captions@aux{#1}{#2}}
3045 \else
3046   \def\bbl@inikv@captions#1#2{%
3047     \bbl@ini@captions@aux{#1}{#2}}
3048 \fi

```

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

```

3049 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3050   \bbl@replace\bbl@tempa{.template}{}%
3051   \def\bbl@toreplace{#1}{}%
3052   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3053   \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3054   \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3055   \bbl@replace\bbl@toreplace{[ ]}{\name\endcsname{}}%
3056   \bbl@replace\bbl@toreplace{[ ]}{\endcsname{}}%
3057   \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3058   \ifin@
3059     \@nameuse{\bbl@patch\bbl@tempa}%
3060     \global\bbl@csarg\let{\bbl@tempa fmt#2}\bbl@toreplace
3061   \fi
3062   \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3063   \ifin@

```

```

3064 \global\bbbl@csarg\let{\bbbl@tempa fmt@#2}\bbbl@toreplace
3065 \bbbl@exp{\gdef\<fnum@\bbbl@tempa>{%
3066   \\bbbl@ifunset{bbbl@\bbbl@tempa fmt@\\languagename}%
3067   {\fnum@\bbbl@tempa}}%
3068   {\@nameuse{bbbl@\bbbl@tempa fmt@\\languagename}}}%
3069 \fi}
3070 \def\bbbl@ini@captions@aux#1#2{%
3071   \bbbl@trim@def\bbbl@tempa{#1}%
3072   \bbbl@xin@{.template}{\bbbl@tempa}%
3073   \ifin@
3074     \bbbl@ini@captions@template{#2}\languagename
3075   \else
3076     \bbbl@ifblank{#2}%
3077     {\bbbl@exp{%
3078       \toks@{\\bbbl@nocaption{\bbbl@tempa}{\languagename\bbbl@tempa name}}}%
3079       {\bbbl@trim\toks@{#2}}}%
3080     \bbbl@exp{%
3081       \\bbbl@add\\bbbl@savestrings{%
3082         \\SetString\<\bbbl@tempa name>{\the\toks@}}%
3083       \toks@\expandafter{\bbbl@captionslist}%
3084       \bbbl@exp{\\in@{\<\bbbl@tempa name>}{\the\toks@}}%
3085       \ifin@\\else
3086         \bbbl@exp{%
3087           \\bbbl@add\<\bbbl@extracaps@\languagename>{\<\bbbl@tempa name>}%
3088           \\bbbl@tglobal\<\bbbl@extracaps@\languagename>}%
3089         \fi
3090       \fi}

```

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

```

3091 \def\bbbl@list@the{%
3092   part,chapter,section,subsection,subsubsection,paragraph,%
3093   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3094   table,page,footnote,mpfootnote,mpfn}
3095 \def\bbbl@map@cnt#1{% #1: roman,etc, // #2:enumi,etc
3096   \bbbl@ifunset{bbbl@map@#1@\languagename}%
3097   {\@nameuse{#1}}%
3098   {\@nameuse{bbbl@map@#1@\languagename}}}
3099 \def\bbbl@inikv@labels#1#2{%
3100   \in@{.map}{#1}%
3101   \ifin@
3102     \ifx\bbbl@KVP@labels\@nnil\else
3103       \bbbl@xin@{ map }{ \bbbl@KVP@labels\space}%
3104     \ifin@
3105       \def\bbbl@tempc{#1}%
3106       \bbbl@replace\bbbl@tempc{.map}{}%
3107       \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3108       \bbbl@exp{%
3109         \gdef\<\bbbl@map@\bbbl@tempc @\languagename>%
3110         {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3111       \bbbl@foreach\bbbl@list@the{%
3112         \bbbl@ifunset{the##1}{%
3113           {\bbbl@exp{\let\\bbbl@tempd\<the##1>}%
3114             \bbbl@exp{%
3115               \\bbbl@sreplace\<the##1>%
3116               {\<\bbbl@tempc>{##1}}{\\bbbl@map@cnt{\bbbl@tempc}{##1}}%
3117               \\bbbl@sreplace\<the##1>%
3118               {\<\@empty @\bbbl@tempc>\<c##1>}{\\bbbl@map@cnt{\bbbl@tempc}{##1}}}%
3119             \expandafter\ifx\csname the##1\endcsname\bbbl@tempd\else
3120             \toks@\expandafter\expandafter\expandafter{%
3121               \csname the##1\endcsname}%
3122             \expandafter\xdef\csname the##1\endcsname{\the\toks@}}%
3123           \fi}}%
3124     \fi}

```

```

3125 \fi
3126 %
3127 \else
3128 %
3129 % The following code is still under study. You can test it and make
3130 % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3131 % language dependent.
3132 \in@{enumerate.}{#1}%
3133 \ifin@
3134 \def\bb@tempa{#1}%
3135 \bb@replace\bb@tempa{enumerate.}{}%
3136 \def\bb@toreplace{#2}%
3137 \bb@replace\bb@toreplace{[ ]}{\nobreakspace{}}%
3138 \bb@replace\bb@toreplace{{}}{\csname the}%
3139 \bb@replace\bb@toreplace{{}}{\endcsname{}}%
3140 \toks@\expandafter{\bb@toreplace}%
3141 % TODO. Execute only once:
3142 \bb@exp{%
3143   \\bb@add\<extras\languagename>{%
3144     \\babel@save\<labelenum\romannumeral\bb@tempa>%
3145     \def\<labelenum\romannumeral\bb@tempa>{\the\toks@}}%
3146   \\bb@togonal\<extras\languagename>}%
3147 \fi
3148 \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.

```

3149 \def\bb@chapttype{chapter}
3150 \ifx\@makechapterhead\@undefined
3151 \let\bb@patchchapter\relax
3152 \else\ifx\thechapter\@undefined
3153 \let\bb@patchchapter\relax
3154 \else\ifx\ps@headings\@undefined
3155 \let\bb@patchchapter\relax
3156 \else
3157 \def\bb@patchchapter{%
3158   \global\let\bb@patchchapter\relax
3159   \gdef\bb@chfmt{%
3160     \bb@ifunset{bb@\bb@chapttype fmt@\languagename}%
3161     {\@chapapp\space\thechapter}
3162     {\@nameuse{bb@\bb@chapttype fmt@\languagename}}}
3163   \bb@add\appendix{\def\bb@chapttype{appendix}}% Not harmful, I hope
3164   \bb@sreplace\ps@headings{\@chapapp\ \thechapter}{\bb@chfmt}%
3165   \bb@sreplace\chaptermark{\@chapapp\ \thechapter}{\bb@chfmt}%
3166   \bb@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bb@chfmt}%
3167   \bb@togonal\appendix
3168   \bb@togonal\ps@headings
3169   \bb@togonal\chaptermark
3170   \bb@togonal\@makechapterhead}
3171 \let\bb@patchappendix\bb@patchchapter
3172 \fi\fi\fi
3173 \ifx\@part\@undefined
3174 \let\bb@patchpart\relax
3175 \else
3176 \def\bb@patchpart{%
3177   \global\let\bb@patchpart\relax
3178   \gdef\bb@partformat{%
3179     \bb@ifunset{bb@partfmt@\languagename}%
3180     {\partname\nobreakspace\thepart}
3181     {\@nameuse{bb@partfmt@\languagename}}}
3182   \bb@sreplace\@part{\partname\nobreakspace\thepart}{\bb@partformat}%

```

```

3183 \bbl@tglobal\@part}
3184 \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

3185 \let\bbl@calendar\@empty
3186 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3187 \def\bbl@localedate#1#2#3#4{%
3188 \begingroup
3189 \edef\bbl@they{#2}%
3190 \edef\bbl@them{#3}%
3191 \edef\bbl@thed{#4}%
3192 \edef\bbl@tempe{%
3193 \bbl@ifunset{bbl@calpr@\languagename}{\bbl@cl{calpr}},%
3194 #1}%
3195 \bbl@replace\bbl@tempe{ }{%
3196 \bbl@replace\bbl@tempe{CONVERT}{convert=% Hackish
3197 \bbl@replace\bbl@tempe{convert}{convert=%}
3198 \let\bbl@ld@calendar\@empty
3199 \let\bbl@ld@variant\@empty
3200 \let\bbl@ld@convert\relax
3201 \def\bbl@tempb##1=##2\@{\@namedef{bbl@ld@##1}{##2}}%
3202 \bbl@foreach\bbl@tempe{\bbl@tempb##1\@}%
3203 \bbl@replace\bbl@ld@calendar{gregorian}{%
3204 \ifx\bbl@ld@calendar\@empty\else
3205 \ifx\bbl@ld@convert\relax\else
3206 \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3207 {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3208 \fi
3209 \fi
3210 \@nameuse{bbl@precalendar}% Remove, eg, +, -civil (-ca-islamic)
3211 \edef\bbl@calendar{% Used in \month..., too
3212 \bbl@ld@calendar
3213 \ifx\bbl@ld@variant\@empty\else
3214 .\bbl@ld@variant
3215 \fi}%
3216 \bbl@cased
3217 {\@nameuse{bbl@date@\languagename @\bbl@calendar}%
3218 \bbl@they\bbl@them\bbl@thed}%
3219 \endgroup}
3220 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3221 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3222 \bbl@trim@def\bbl@tempa{#1.#2}%
3223 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3224 {\bbl@trim@def\bbl@tempa{#3}%
3225 \bbl@trim\toks@{#5}%
3226 \@temptokena\expandafter{\bbl@savedate}%
3227 \bbl@exp{% Reverse order - in ini last wins
3228 \def\\bbl@savedate{%
3229 \\SetString<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3230 \the\@temptokena}}%
3231 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3232 {\lowercase{\def\bbl@tempb{#6}}}%
3233 \bbl@trim@def\bbl@toreplace{#5}%
3234 \bbl@TG@@date
3235 \global\bbl@csarg\let{date@\languagename @\bbl@tempb}\bbl@toreplace
3236 \ifx\bbl@savetoday\@empty
3237 \bbl@exp{% TODO. Move to a better place.
3238 \\AfterBabelCommands{%
3239 \def<\languagename date>{\\protect<\languagename date >}%
3240 \\newcommand<\languagename date >[4][{%
3241 \\bbl@usedategroupttrue
3242 \<bbl@ensure@\languagename>{%

```

```

3243         \\localedate{####1}{####2}{####3}{####4}}}%
3244     \def\\bbl@savetoday{%
3245         \\SetString\\today{%
3246             <\language name date>[convert]%
3247             {\\the\year}{\\the\month}{\\the\day}}}%
3248     \fi}%
3249     {}}}

```

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).

```

3250 \let\bbl@calendar\@empty
3251 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3252     \@nameuse{\bbl@ca@#2}#1\@@}
3253 \newcommand\babelDateSpace{\nobreakspace}
3254 \newcommand\babelDateDot{.\@} % TODO. \let instead of repeating
3255 \newcommand\babelDated[1]{\number#1}
3256 \newcommand\babelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3257 \newcommand\babelDateM[1]{\number#1}
3258 \newcommand\babelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3259 \newcommand\babelDateMMM[1]{%
3260     \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3261 \newcommand\babelDatey[1]{\number#1}%
3262 \newcommand\babelDateyy[1]{%
3263     \ifnum#1<10 0\number#1 %
3264     \else\ifnum#1<100 \number#1 %
3265     \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3266     \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3267     \else
3268         \bbl@error{limit-two-digits}{\number#1}%
3269     \fi\fi\fi\fi}
3270 \newcommand\babelDateyyyy[1]{\number#1} % TODO - add leading 0
3271 \newcommand\babelDateU[1]{\number#1}%
3272 \def\bbl@replace@finish@iii#1{%
3273     \bbl@exp{\def\#1####1####2####3{\the\toks@}}
3274 \def\bbl@TG@date{%
3275     \bbl@replace\bbl@toreplace{[ ]}{\babelDateSpace}}%
3276     \bbl@replace\bbl@toreplace{[.]}{\babelDateDot}}%
3277     \bbl@replace\bbl@toreplace{[d]}{\babelDated{####3}}%
3278     \bbl@replace\bbl@toreplace{[dd]}{\babelDatedd{####3}}%
3279     \bbl@replace\bbl@toreplace{[M]}{\babelDateM{####2}}%
3280     \bbl@replace\bbl@toreplace{[MM]}{\babelDateMM{####2}}%
3281     \bbl@replace\bbl@toreplace{[MMM]}{\babelDateMMM{####2}}%
3282     \bbl@replace\bbl@toreplace{[y]}{\babelDatey{####1}}%
3283     \bbl@replace\bbl@toreplace{[yy]}{\babelDateyy{####1}}%
3284     \bbl@replace\bbl@toreplace{[yyy]}{\babelDateyyy{####1}}%
3285     \bbl@replace\bbl@toreplace{[U]}{\babelDateU{####1}}%
3286     \bbl@replace\bbl@toreplace{[y]}{\bbl@datecptr[####1]}%
3287     \bbl@replace\bbl@toreplace{[U]}{\bbl@datecptr[####1]}%
3288     \bbl@replace\bbl@toreplace{[m]}{\bbl@datecptr[####2]}%
3289     \bbl@replace\bbl@toreplace{[d]}{\bbl@datecptr[####3]}%
3290     \bbl@replace@finish@iii\bbl@toreplace}
3291 \def\bbl@datecptr\expandafter\bbl@xdatecptr\expandafter}
3292 \def\bbl@xdatecptr[#1|#2]{\localenumeral{#2}{#1}}

```

Transforms.

```

3293 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3294 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3295 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3296     #1[#2]{#3}{#4}{#5}}
3297 \begingroup % A hack. TODO. Don't require a specific order
3298     \catcode`\%=12

```



```

3299 \catcode`\&=14
3300 \gdef\bb@transforms#1#2#3{&%
3301 \directlua{
3302     local str = [=[#2]=]
3303     str = str:gsub('%.%d+%.%d+$', '')
3304     token.set_macro('babeltempa', str)
3305 }&%
3306 \def\babeltempc{}&%
3307 \bb@xin@{,\babeltempa,}{,\bb@KVP@transforms,}&%
3308 \ifin@ \else
3309     \bb@xin@{: \babeltempa,}{,\bb@KVP@transforms,}&%
3310 \fi
3311 \ifin@
3312     \bb@foreach\bb@KVP@transforms{&%
3313         \bb@xin@{: \babeltempa,}{, #1,}&%
3314         \ifin@ &% font:font:transform syntax
3315             \directlua{
3316                 local t = {}
3317                 for m in string.gmatch('#1..' ':', '(.-):') do
3318                     table.insert(t, m)
3319                 end
3320                 table.remove(t)
3321                 token.set_macro('babeltempc', ', fonts=' .. table.concat(t, ' '))
3322             }&%
3323         \fi}&%
3324 \in@{.0$}{#2$}&%
3325 \ifin@
3326     \directlua{&% (\attribute) syntax
3327         local str = string.match([[ \bb@KVP@transforms]],
3328             '%(([^%(-)]^)[^)]-\babeltempa')
3329         if str == nil then
3330             token.set_macro('babeltempb', '')
3331         else
3332             token.set_macro('babeltempb', ', attribute=' .. str)
3333         end
3334     }&%
3335 \toks@{#3}&%
3336 \bb@exp{&%
3337     \\g@addto@macro\\bb@release@transforms{&%
3338     \relax &% Closes previous \bb@transforms@aux
3339     \\bb@transforms@aux
3340     \\#1{label=\babeltempa\babeltempb\babeltempc}&%
3341     {\language\the\toks@}}&%
3342 \else
3343     \g@addto@macro\bb@release@transforms{, {#3}}&%
3344 \fi
3345 \fi}
3346 \endgroup

```

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

```

3347 \def\bb@provide@lsys#1{%
3348     \bb@ifunset{bb@lname@#1}%
3349     {\bb@load@info{#1}}%
3350     {%
3351     \bb@csarg\let{lsys@#1}\@empty
3352     \bb@ifunset{bb@sname@#1}{\bb@csarg\gdef{sname@#1}{Default}}{}%
3353     \bb@ifunset{bb@sotf@#1}{\bb@csarg\gdef{sotf@#1}{DFLT}}{}%
3354     \bb@csarg\bb@add@list{lsys@#1}{Script=\bb@cs{sname@#1}}%
3355     \bb@ifunset{bb@lname@#1}{%
3356         {\bb@csarg\bb@add@list{lsys@#1}{Language=\bb@cs{lname@#1}}}%
3357     \ifcase\bb@engine\or\or
3358     \bb@ifunset{bb@prehc@#1}{%

```

```

3359     {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3360     }%
3361     {\ifx\bbl@xenohyph\undefined
3362     \global\let\bbl@xenohyph\bbl@xenohyph@d
3363     \ifx\AtBeginDocument\@notprerr
3364     \expandafter\@secondoftwo % to execute right now
3365     \fi
3366     \AtBeginDocument{%
3367     \bbl@patchfont{\bbl@xenohyph}%
3368     {\expandafter\select@language\expandafter{\language}}}%
3369     \fi}}%
3370 \fi
3371 \bbl@csarg\bbl@tglobal{\sys@#1}}
3372 \def\bbl@xenohyph@d{%
3373 \bbl@ifset{bbl@prehc@\language}%
3374 {\ifnum\hyphenchar\font=\defaulthyphenchar
3375 \iffontchar\font\bbl@cl{prehc}\relax
3376 \hyphenchar\font\bbl@cl{prehc}\relax
3377 \else\iffontchar\font"200B
3378 \hyphenchar\font"200B
3379 \else
3380 \bbl@warning
3381 {Neither 0 nor ZERO WIDTH SPACE are available\\%
3382 in the current font, and therefore the hyphen\\%
3383 will be printed. Try changing the fontspec's\\%
3384 'HyphenChar' to another value, but be aware\\%
3385 this setting is not safe (see the manual).\\%
3386 Reported}%
3387 \hyphenchar\font\defaulthyphenchar
3388 \fi\fi
3389 \fi}%
3390 {\hyphenchar\font\defaulthyphenchar}}
3391 % \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).

```

3392 \def\bbl@load@info#1{%
3393 \def\BabelBeforeIni##1##2{%
3394 \begingroup
3395 \bbl@read@ini{##1}0%
3396 \endinput % babel- .tex may contain only preamble's
3397 \endgroup}% boxed, to avoid extra spaces:
3398 {\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.

```

3399 \def\bbl@setdigits#1#2#3#4#5{%
3400 \bbl@exp{%
3401 \def<\language digits>###1{% ie, \langdigits
3402 <\bbl@digits@\language>###1\\\@nil}%
3403 \let<\bbl@cntr@digits@\language><\language digits>%
3404 \def<\language counter>###1{% ie, \langcounter
3405 \\expandafter\bbl@counter@\language>%
3406 \\cscname c@###1\endcscname}%
3407 \def<\bbl@counter@\language>###1{% ie, \bbl@counter@lang
3408 \\expandafter\bbl@digits@\language>%
3409 \\number###1\\\@nil}}%
3410 \def\bbl@tempa##1##2##3##4##5{%
3411 \bbl@exp{% Wow, quite a lot of hashes! :- (
3412 \def<\bbl@digits@\language>#####1{%
3413 \\ifx#####1\\\@nil % ie, \bbl@digits@lang

```

```

3414     \\\else
3415         \\\ifx0#####1#1%
3416         \\\else\\\ifx1#####1#2%
3417         \\\else\\\ifx2#####1#3%
3418         \\\else\\\ifx3#####1#4%
3419         \\\else\\\ifx4#####1#5%
3420         \\\else\\\ifx5#####1#1%
3421         \\\else\\\ifx6#####1#2%
3422         \\\else\\\ifx7#####1#3%
3423         \\\else\\\ifx8#####1#4%
3424         \\\else\\\ifx9#####1#5%
3425         \\\else#####1%
3426         \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
3427         \\\expandafter\<bbl@digits@\language\>%
3428     \\\fi}}}%
3429 \bbl@tempa}

```

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

```

3430 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={ }
3431 \ifx\#1%           % \ before, in case #1 is multiletter
3432   \bbl@exp{%
3433     \def\#1\bbl@tempa#####1{%
3434       \<ifcase>#####1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
3435   \else
3436     \toks@\expandafter{\the\toks@\or #1}%
3437     \expandafter\bbl@buildifcase
3438   \fi}

```

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

```

3439 \newcommand\localenumerical[2]{\bbl@cs{cntr@#1@\language}\{#2}}
3440 \def\bbl@localecntr#1#2{\localenumerical{#2}{#1}}
3441 \newcommand\localecounter[2]{%
3442   \expandafter\bbl@localecntr
3443   \expandafter{number\csname c@#2\endcsname}\{#1}}
3444 \def\bbl@alphnumerical#1#2{%
3445   \expandafter\bbl@alphnumerical@i\number#2 76543210\@@{#1}}
3446 \def\bbl@alphnumerical@i#1#2#3#4#5#6#7#8\@@#9{%
3447   \ifcase\@car#8\@nil\or % Currently <10000, but prepared for bigger
3448     \bbl@alphnumerical@ii{#9}000000#1\or
3449     \bbl@alphnumerical@ii{#9}00000#1#2\or
3450     \bbl@alphnumerical@ii{#9}0000#1#2#3\or
3451     \bbl@alphnumerical@ii{#9}000#1#2#3#4\else
3452     \bbl@alphnum@invalid{>9999}%
3453   \fi}
3454 \def\bbl@alphnumerical@ii#1#2#3#4#5#6#7#8{%
3455   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\language}%
3456     {\bbl@cs{cntr@#1.4@\language}\{#5}}
3457     {\bbl@cs{cntr@#1.3@\language}\{#6}}
3458     {\bbl@cs{cntr@#1.2@\language}\{#7}}
3459     {\bbl@cs{cntr@#1.1@\language}\{#8}}
3460   \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3461     \bbl@ifunset{bbl@cntr@#1.S.321@\language}\{#6}}
3462     {\bbl@cs{cntr@#1.S.321@\language}\{#6}}
3463   \fi}%
3464   {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}\{#6}}
3465 \def\bbl@alphnum@invalid#1{%
3466   \bbl@error{alphabetic-too-large}\{#1}\{}}

```

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

```

3467 \def\bbbl@localeinfo#1#2{%
3468   \bbbl@ifunset{bbbl@info@#2}{#1}%
3469     {\bbbl@ifunset{bbbl@csname bbl@info@#2\endcsname @\languagename}{#1}%
3470       {\bbbl@cs{\csname bbl@info@#2\endcsname @\languagename}}}}
3471 \newcommand\localeinfo[1]{%
3472   \ifx*#1\@empty   % TODO. A bit hackish to make it expandable.
3473     \bbbl@afterelse\bbbl@localeinfo{%
3474   \else
3475     \bbbl@localeinfo
3476       {\bbbl@error{no-ini-info}{}}{}}}%
3477   {#1}%
3478 \fi}
3479 % \@namedef{bbbl@info@name.locale}{lcname}
3480 \@namedef{bbbl@info@tag.ini}{lini}
3481 \@namedef{bbbl@info@name.english}{elname}
3482 \@namedef{bbbl@info@name.opentype}{lname}
3483 \@namedef{bbbl@info@tag.bcp47}{tbcpl}
3484 \@namedef{bbbl@info@language.tag.bcp47}{lbcpl}
3485 \@namedef{bbbl@info@tag.opentype}{lotf}
3486 \@namedef{bbbl@info@script.name}{esname}
3487 \@namedef{bbbl@info@script.name.opentype}{sname}
3488 \@namedef{bbbl@info@script.tag.bcp47}{sbcp}
3489 \@namedef{bbbl@info@script.tag.opentype}{sotf}
3490 \@namedef{bbbl@info@region.tag.bcp47}{rbcp}
3491 \@namedef{bbbl@info@variant.tag.bcp47}{vbcpl}
3492 \@namedef{bbbl@info@extension.t.tag.bcp47}{extt}
3493 \@namedef{bbbl@info@extension.u.tag.bcp47}{extu}
3494 \@namedef{bbbl@info@extension.x.tag.bcp47}{extx}

```

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

```

3495 \ifcase\bbbl@engine % Converts utf8 to its code (expandable)
3496   \def\bbbl@uftocode#1{\the\numexpr\decode@UTFviii#1\relax}
3497 \else
3498   \def\bbbl@uftocode#1{\expandafter`\string#1}
3499 \fi
3500 % Still somewhat hackish. WIP. Note |\str_if_eq:nnTF| is fully
3501 % expandable (|\bbbl@ifsamestring| isn't).
3502 \providecommand\BCPdata{}
3503 \ifx\renewcommand\undefined\else % For plain. TODO. It's a quick fix
3504   \renewcommand\BCPdata[1]{\bbbl@bcpdata@i#1\@empty}
3505   \def\bbbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3506     \nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3507     {\bbbl@bcpdata@ii{#6}\bbbl@main@language}%
3508     {\bbbl@bcpdata@ii{#1#2#3#4#5#6}\languagename}}%
3509   \def\bbbl@bcpdata@ii#1#2{%
3510     \bbbl@ifunset{bbbl@info@#1.tag.bcp47}%
3511       {\bbbl@error{unknown-ini-field}{#1}}{}}%
3512     {\bbbl@ifunset{bbbl@csname bbl@info@#1.tag.bcp47\endcsname @#2}{}}%
3513     {\bbbl@cs{\csname bbl@info@#1.tag.bcp47\endcsname @#2}}}}
3514 \fi
3515 \@namedef{bbbl@info@casing.tag.bcp47}{casing}
3516 \newcommand\BabelUppercaseMapping[3]{%
3517   \DeclareUppercaseMapping[\nameuse{bbbl@casing@#1}]{#2}{#3}}
3518 \newcommand\BabelTitlecaseMapping[3]{%
3519   \DeclareTitlecaseMapping[\nameuse{bbbl@casing@#1}]{#2}{#3}}
3520 \newcommand\BabelLowercaseMapping[3]{%
3521   \DeclareLowercaseMapping[\nameuse{bbbl@casing@#1}]{#2}{#3}}

```

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

```

3522 \def\bbbl@casemapping#1#2#3{% 1:variant
3523   \def\bbbl@tempa##1 ##2{% Loop
3524     \bbbl@casemapping@i{##1}%

```

```

3525 \ifx\@empty##2\else\bbbl@afterfi\bbbl@tempa##2\fi}%
3526 \edef\bbbl@templ{\@nameuse{bbbl@casing@#2}#1}% Language code
3527 \def\bbbl@tempe{0}% Mode (upper/lower...)
3528 \def\bbbl@tempc{#3}% Casing list
3529 \expandafter\bbbl@tempa\bbbl@tempc\@empty}
3530 \def\bbbl@casemapping@i#1{%
3531 \def\bbbl@tempb{#1}%
3532 \ifcase\bbbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
3533 \@nameuse{regex_replace_all:nnN}%
3534 {[\x{c0}-\x{ff}][\x{80}-\x{bf}]*}{\0}\bbbl@tempb
3535 \else
3536 \@nameuse{regex_replace_all:nnN}{.}{\0}\bbbl@tempb % TODO. needed?
3537 \fi
3538 \expandafter\bbbl@casemapping@ii\bbbl@tempb@@}
3539 \def\bbbl@casemapping@ii#1#2#3\@@{%
3540 \in@{#1#3}{<>}% ie, if <u>, <l>, <t>
3541 \ifin@
3542 \edef\bbbl@tempe{%
3543 \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
3544 \else
3545 \ifcase\bbbl@tempe\relax
3546 \DeclareUppercaseMapping[\bbbl@templ]{\bbbl@uftocode{#1}}{#2}%
3547 \DeclareLowercaseMapping[\bbbl@templ]{\bbbl@uftocode{#2}}{#1}%
3548 \or
3549 \DeclareUppercaseMapping[\bbbl@templ]{\bbbl@uftocode{#1}}{#2}%
3550 \or
3551 \DeclareLowercaseMapping[\bbbl@templ]{\bbbl@uftocode{#1}}{#2}%
3552 \or
3553 \DeclareTitlecaseMapping[\bbbl@templ]{\bbbl@uftocode{#1}}{#2}%
3554 \fi
3555 \fi}

```

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

```

3556 <<{*More package options}>> ≡
3557 \DeclareOption{ensureinfo=off}{}
3558 <</More package options>>
3559 \let\bbbl@ensureinfo\@gobble
3560 \newcommand\BabelEnsureInfo{%
3561 \ifx\InputIfFileExists\@undefined\else
3562 \def\bbbl@ensureinfo##1{%
3563 \bbbl@ifunset{bbbl@lname@##1}{\bbbl@load@info{##1}}{}}%
3564 \fi
3565 \bbbl@foreach\bbbl@loaded{%
3566 \let\bbbl@ensuring\@empty % Flag used in a couple of babel-*.tex files
3567 \def\languagename{##1}%
3568 \bbbl@ensureinfo{##1}}}%
3569 \ifpackagewith{babel}{ensureinfo=off}{}%
3570 {\AtEndOfPackage{% Test for plain.
3571 \ifx\@undefined\bbbl@loaded\else\BabelEnsureInfo\fi}}

```

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

```

3572 \newcommand\getlocaleproperty{%
3573 \@ifstar\bbbl@getproperty@s\bbbl@getproperty@x}
3574 \def\bbbl@getproperty@s#1#2#3{%
3575 \let#1\relax
3576 \def\bbbl@elt##1##2##3{%
3577 \bbbl@ifsamestring{##1/##2}{##3}%
3578 {\providecommand#1{##3}%
3579 \def\bbbl@elt####1####2####3{}}}%
3580 {}}%
3581 \bbbl@cs{inidata@#2}}%
3582 \def\bbbl@getproperty@x#1#2#3{%

```

```

3583 \bbl@getproperty@{#1}{#2}{#3}%
3584 \ifx#1\relax
3585 \bbl@error{unknown-locale-key}{#1}{#2}{#3}%
3586 \fi}
3587 \let\bbl@ini@loaded\@empty
3588 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
3589 \def\ShowLocaleProperties#1{%
3590 \typeout{}%
3591 \typeout{*** Properties for language '#1' ***}
3592 \def\bbl@elt##1##2##3{\typeout{##1/##2 = ##3}}%
3593 \@nameuse{\bbl@inidata@#1}%
3594 \typeout{*****}}

```

5 Adjusting the Babel behavior

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

```

3595 \newcommand\babeladjust[1]{% TODO. Error handling.
3596 \bbl@forkv{#1}{%
3597 \bbl@ifunset{\bbl@ADJ@##1@##2}%
3598 {\bbl@cs{ADJ@##1}{##2}}%
3599 {\bbl@cs{ADJ@##1@##2}}}
3600 %
3601 \def\bbl@adjust@lua#1#2{%
3602 \ifvmode
3603 \ifnum\currentgrouplevel=\z@
3604 \directlua{ Babel.#2 }%
3605 \expandafter\expandafter\expandafter@gobble
3606 \fi
3607 \fi
3608 {\bbl@error{adjust-only-vertical}{#1}{}}% Gobbled if everything went ok.
3609 \@namedef{\bbl@ADJ@bidi.mirroring@on}{%
3610 \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3611 \@namedef{\bbl@ADJ@bidi.mirroring@off}{%
3612 \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3613 \@namedef{\bbl@ADJ@bidi.text@on}{%
3614 \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3615 \@namedef{\bbl@ADJ@bidi.text@off}{%
3616 \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3617 \@namedef{\bbl@ADJ@bidi.math@on}{%
3618 \let\bbl@noamsmath\@empty}
3619 \@namedef{\bbl@ADJ@bidi.math@off}{%
3620 \let\bbl@noamsmath\relax}
3621 %
3622 \@namedef{\bbl@ADJ@bidi.mapdigits@on}{%
3623 \bbl@adjust@lua{bidi}{digits_mapped=true}}
3624 \@namedef{\bbl@ADJ@bidi.mapdigits@off}{%
3625 \bbl@adjust@lua{bidi}{digits_mapped=false}}
3626 %
3627 \@namedef{\bbl@ADJ@linebreak.sea@on}{%
3628 \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3629 \@namedef{\bbl@ADJ@linebreak.sea@off}{%
3630 \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3631 \@namedef{\bbl@ADJ@linebreak.cjk@on}{%
3632 \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3633 \@namedef{\bbl@ADJ@linebreak.cjk@off}{%
3634 \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3635 \@namedef{\bbl@ADJ@justify.arabic@on}{%
3636 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3637 \@namedef{\bbl@ADJ@justify.arabic@off}{%
3638 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3639 %
3640 \def\bbl@adjust@layout#1{%

```

```

3641 \ifvmode
3642   #1%
3643   \expandafter\@gobble
3644 \fi
3645 {\bbl@error{layout-only-vertical}{}}{}% Gobbled if everything went ok.
3646 \@namedef{bbl@ADJ@layout.tabular@on}{%
3647   \ifnum\bbl@tabular@mode=\tw@
3648     \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}%
3649   \else
3650     \chardef\bbl@tabular@mode\@ne
3651   \fi}
3652 \@namedef{bbl@ADJ@layout.tabular@off}{%
3653   \ifnum\bbl@tabular@mode=\tw@
3654     \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}%
3655   \else
3656     \chardef\bbl@tabular@mode\z@
3657   \fi}
3658 \@namedef{bbl@ADJ@layout.lists@on}{%
3659   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3660 \@namedef{bbl@ADJ@layout.lists@off}{%
3661   \bbl@adjust@layout{\let\list\bbl@OL@list}}
3662 %
3663 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3664   \bbl@bcppallowedtrue}
3665 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3666   \bbl@bcppallowedfalse}
3667 \@namedef{bbl@ADJ@autoload.bcp47.prefix#1}{%
3668   \def\bbl@bcp@prefix{#1}}
3669 \def\bbl@bcp@prefix{bcp47-}
3670 \@namedef{bbl@ADJ@autoload.options#1}{%
3671   \def\bbl@autoload@options{#1}}
3672 \let\bbl@autoload@bcptoptions\@empty
3673 \@namedef{bbl@ADJ@autoload.bcp47.options#1}{%
3674   \def\bbl@autoload@bcptoptions{#1}}
3675 \newif\ifbbl@bcptoname
3676 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3677   \bbl@bcptonametrue}
3678 \BabelEnsureInfo
3679 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3680   \bbl@bcptonamefalse}
3681 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3682   \directlua{ Babel.ignore_pre_char = function(node)
3683     return (node.lang == \the\csname l@nohyphenation\endcsname)
3684   end }}
3685 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3686   \directlua{ Babel.ignore_pre_char = function(node)
3687     return false
3688   end }}
3689 \@namedef{bbl@ADJ@interchar.disable@nohyphenation}{%
3690   \def\bbl@ignoreinterchar{%
3691     \ifnum\language=\l@nohyphenation
3692       \expandafter\@gobble
3693     \else
3694       \expandafter\@firstofone
3695     \fi}}
3696 \@namedef{bbl@ADJ@interchar.disable@off}{%
3697   \let\bbl@ignoreinterchar\@firstofone}
3698 \@namedef{bbl@ADJ@select.write@shift}{%
3699   \let\bbl@restorelastskip\relax
3700   \def\bbl@savelastskip{%
3701     \let\bbl@restorelastskip\relax
3702   \ifvmode
3703     \ifdim\lastskip=\z@

```

```

3704     \let\bbl@restorelastskip\nobreak
3705     \else
3706     \bbl@exp{%
3707     \def\\bbl@restorelastskip{%
3708     \skip@=\the\lastskip
3709     \\nobreak \vskip-\skip@ \vskip\skip@}}%
3710     \fi
3711     \fi}}
3712 \@namedef{bbl@ADJ@select.write@keep}{%
3713     \let\bbl@restorelastskip\relax
3714     \let\bbl@savelastskip\relax}
3715 \@namedef{bbl@ADJ@select.write@omit}{%
3716     \AddBabelHook{babel-select}{beforestart}{%
3717     \expandafter\babel@aux\expandafter{\bbl@main@language}}}%
3718     \let\bbl@restorelastskip\relax
3719     \def\bbl@savelastskip##1\bbl@restorelastskip{}}
3720 \@namedef{bbl@ADJ@select.encoding@off}{%
3721     \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.

```

3722 <<{*More package options}>> ≡
3723 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3724 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3725 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3726 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3727 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3728 <</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.

```

3729 \bbl@trace{Cross referencing macros}
3730 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3731     \def\@newl@bel#1#2#3{%
3732     {\@safe@activestrue
3733     \bbl@ifunset{#1@#2}%
3734     \relax
3735     \gdef\@multiplelabels{%
3736     \@latex@warning@no@line{There were multiply-defined labels}}%
3737     \@latex@warning@no@line{Label `#2' multiply defined}}%
3738     \global\@namedef{#1@#2}{#3}}

```

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

```

3739 \CheckCommand*\@testdef[3]{%
3740     \def\reserved@a{#3}%
3741     \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3742     \else
3743     \@tempwattrue
3744     \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.

```

3745 \def\@testdef#1#2#3{% TODO. With @samestring?
3746   \@safe@activestru
3747   \expandafter\let\expandafter\bbl@tempa\csname #1#2\endcsname
3748   \def\bbl@tempb{#3}%
3749   \@safe@activesfalse
3750   \ifx\bbl@tempa\relax
3751   \else
3752     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3753   \fi
3754   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3755   \ifx\bbl@tempa\bbl@tempb
3756   \else
3757     \@tempwatru
3758   \fi}
3759 \fi

```

`\ref` The same holds for the macro `\ref` that references a label and `\pageref` to reference a page. We `\pageref` make them robust as well (if they weren't already) to prevent problems if they should become expanded at the wrong moment.

```

3760 \bbl@xin@{R}\bbl@opt@safe
3761 \ifin@
3762 \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3763 \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3764   {\expandafter\strip@prefix\meaning\ref}%
3765 \ifin@
3766   \bbl@redefine\@kernel@ref#1{%
3767     \@safe@activestru\org@@kernel@ref{#1}\@safe@activesfalse}
3768   \bbl@redefine\@kernel@pageref#1{%
3769     \@safe@activestru\org@@kernel@pageref{#1}\@safe@activesfalse}
3770   \bbl@redefine\@kernel@sref#1{%
3771     \@safe@activestru\org@@kernel@sref{#1}\@safe@activesfalse}
3772   \bbl@redefine\@kernel@spageref#1{%
3773     \@safe@activestru\org@@kernel@spageref{#1}\@safe@activesfalse}
3774   \else
3775     \bbl@redefinero\ref#1{%
3776       \@safe@activestru\org@ref{#1}\@safe@activesfalse}
3777     \bbl@redefinero\pageref#1{%
3778       \@safe@activestru\org@pageref{#1}\@safe@activesfalse}
3779   \fi
3780 \else
3781   \let\org@ref\ref
3782   \let\org@pageref\pageref
3783 \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.

```

3784 \bbl@xin@{B}\bbl@opt@safe
3785 \ifin@
3786   \bbl@redefine\@citex[#1]#2{%
3787     \@safe@activestru\edef\bbl@tempa{#2}\@safe@activesfalse
3788     \org@@citex[#1]{\bbl@tempa}}

```

Unfortunately, the packages `natbib` and `cite` need a different definition of `\@citex`... To begin with, `natbib` has a definition for `\@citex` with *three* arguments... We only know that a package is loaded when `\begin{document}` is executed, so we need to postpone the different redefinition.

```

3789 \AtBeginDocument{%
3790   \ifpackageloaded{natbib}{%

```

Notice that we use `\def` here instead of `\bb@l@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.)

```

3791   \def\@citex[#1][#2]#3{%
3792     \@safe@activestrue\edef\bb@tempa{#3}\@safe@activestrue
3793     \org@citex[#1][#2]{\bb@tempa}}%
3794   }{}}

```

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

```

3795 \AtBeginDocument{%
3796   \ifpackageloaded{cite}{%
3797     \def\@citex[#1]#2{%
3798       \@safe@activestrue\org@citex[#1][#2]\@safe@activestrue}%
3799     }{}}

```

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

```

3800 \bb@l@redefine\nocite#1{%
3801   \@safe@activestrue\org@nocite{#1}\@safe@activestrue}

```

`\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@activestrue` 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 `\bb@l@cite@choice` to select the proper definition for `\bibcite`. This new definition is then activated.

```

3802 \bb@l@redefine\bibcite{%
3803   \bb@l@cite@choice
3804   \bibcite}

```

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

```

3805 \def\bb@l@bibcite#1#2{%
3806   \org@bibcite{#1}{\@safe@activestrue#2}}

```

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

```

3807 \def\bb@l@cite@choice{%
3808   \global\let\bibcite\bb@l@bibcite
3809   \ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
3810   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{%
3811   \global\let\bibcite\relax}

```

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

```

3812 \AtBeginDocument{\bb@l@cite@choice}

```

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

```

3813 \bb@l@redefine\@bibitem#1{%
3814   \@safe@activestrue\org@bibitem{#1}\@safe@activestrue}
3815 \else
3816   \let\org@nocite\nocite
3817   \let\org@citex\@citex
3818   \let\org@bibcite\bibcite
3819   \let\org@bibitem\@bibitem
3820 \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.

```
3821 \bbl@trace{Marks}
3822 \IfBabelLayout{sectioning}
3823   {\ifx\bbl@opt@headfoot\@nnil
3824     \g@addto@macro\@resetactivechars{%
3825       \set@typeset@protect
3826       \expandafter\select@language@x\expandafter{\bbl@main@language}%
3827       \let\protect\noexpand
3828       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3829         \edef\thepage{%
3830           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3831       \fi}%
3832   \fi}
3833 {\ifbbl@single\else
3834   \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
3835   \markright#1{%
3836     \bbl@ifblank{#1}%
3837     {\org@markright{}}%
3838     {\toks@{#1}%
3839     \bbl@exp{%
3840       \\org@markright{\\protect\\foreignlanguage{\language}%
3841         {\\protect\\bbl@restore@actives\the\toks@}}}}%
3842
```

`\markboth` The definition of `\markboth` is equivalent to that of `\markright`, except that we need two token registers. The documentclasses report and book define and set the headings for the page. While doing so they also store a copy of `\markboth` in `\@mkboth`. Therefore we need to check whether `\@mkboth` has already been set. If so we need to do that again with the new definition of `\markboth`. (As of Oct 2019, \LaTeX stores the definition in an intermediate macro, so it's not necessary anymore, but it's preserved for older versions.)

```
3842   \ifx\@mkboth\markboth
3843     \def\bbl@tempc{\let\@mkboth\markboth}%
3844   \else
3845     \def\bbl@tempc{%
3846     \fi
3847     \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
3848     \markboth#1#2{%
3849       \protected@edef\bbl@tempb##1{%
3850         \protect\foreignlanguage
3851         {\language}{\protect\bbl@restore@actives##1}}%
3852       \bbl@ifblank{#1}%
3853       {\toks@{}}%
3854       {\toks@\expandafter{\bbl@tempb{#1}}}%
3855       \bbl@ifblank{#2}%
3856       {\@temptokena{}}%
3857       {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3858       \bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}}%
3859       \bbl@tempc
3860     \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.

```

3861 \bbl@trace{Preventing clashes with other packages}
3862 \ifx\org@ref\undefined\else
3863   \bbl@xin@{R}\bbl@opt@safe
3864   \ifin@
3865     \AtBeginDocument{%
3866       \ifpackageloaded{ifthen}{%
3867         \bbl@redefine@long\ifthenelse#1#2#3{%
3868           \let\bbl@temp@pref\pageref
3869           \let\pageref\org@pageref
3870           \let\bbl@temp@ref\ref
3871           \let\ref\org@ref
3872           \@safe@activestru
3873           \org@ifthenelse{#1}%
3874             {\let\pageref\bbl@temp@pref
3875              \let\ref\bbl@temp@ref
3876              \@safe@activesfalse
3877              #2}%
3878             {\let\pageref\bbl@temp@pref
3879              \let\ref\bbl@temp@ref
3880              \@safe@activesfalse
3881              #3}%
3882           }%
3883         }{}%
3884       }
3885 \fi

```

5.3.2 varioref

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

```

3886 \AtBeginDocument{%
3887   \ifpackageloaded{varioref}{%
3888     \bbl@redefine@\@vppageref#1[#2]#3{%
3889       \@safe@activestru
3890       \org@\@vppageref{#1}[#2]{#3}%
3891       \@safe@activesfalse}%
3892     \bbl@redefine\vrefpagemum#1#2{%
3893       \@safe@activestru
3894       \org@vrefpagemum{#1}{#2}%
3895       \@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.

```

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

```

```

3898     }{}%
3899   }
3900 \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.

```

3901 \AtEndOfPackage{%
3902   \AtBeginDocument{%
3903     \ifpackageloaded{hhline}%
3904       {\expandafter\ifx\csname normal@char\string:\endcsname\relax
3905         \else
3906           \makeatletter
3907           \def\currname{hhline}\input{hhline.sty}\makeatother
3908           \fi}%
3909     {}}

```

`\substitutefontfamily` *Deprecated*. Use the tools provided by \TeX (`\DeclareFontFamilySubstitution`). 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.

```

3910 \def\substitutefontfamily#1#2#3{%
3911   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3912   \immediate\write15{%
3913     \string\ProvidesFile{#1#2.fd}%
3914     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3915     \space generated font description file]^^J
3916     \string\DeclareFontFamily{#1}{#2}{^^J
3917     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
3918     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
3919     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
3920     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
3921     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
3922     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
3923     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
3924     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
3925     }%
3926   \closeout15
3927   }
3928 \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
3929 \bbl@trace{Encoding and fonts}
3930 \newcommand\BabelNonASCII{LGR, LGI, X2, OT2, OT3, OT6, LHE, LWN, LMA, LMC, LMS, LMU}
3931 \newcommand\BabelNonText{TS1, T3, TS3}
3932 \let\org@TeX\TeX
3933 \let\org@LaTeX\LaTeX
3934 \let\ensureascii\@firstofone
3935 \let\asciientoding\@empty
3936 \AtBeginDocument{%
3937   \def\elt#1{, #1,}%

```

```

3938 \edef\bb@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3939 \let\@elt\relax
3940 \let\bb@tempb\@empty
3941 \def\bb@tempc{OT1}%
3942 \bb@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3943   \bb@ifunset{T@#1}{\def\bb@tempb{#1}}}%
3944 \bb@foreach\bb@tempa{%
3945   \bb@xin@{,#1,}{,\BabelNonASCII,}%
3946   \ifin@
3947     \def\bb@tempb{#1}% Store last non-ascii
3948   \else\bb@xin@{,#1,}{,\BabelNonText,}% Pass
3949     \ifin@else
3950       \def\bb@tempc{#1}% Store last ascii
3951     \fi
3952   \fi}%
3953 \ifx\bb@tempb\@empty\else
3954   \bb@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3955   \ifin@else
3956     \edef\bb@tempc{\cf@encoding}% The default if ascii wins
3957   \fi
3958   \let\asciencoding\bb@tempc
3959   \renewcommand\ensureascii[1]{%
3960     {\fontencoding{\asciencoding}\selectfont#1}}%
3961   \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3962   \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3963   \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.

```

3964 \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.

```

3965 \AtBeginDocument{%
3966   \ifpackageloaded{fontspec}%
3967     {\xdef\latinencoding{%
3968       \ifx\UTFencname\undefined
3969         EU\ifcase\bb@engine\or2\or1\fi
3970       \else
3971         \UTFencname
3972       \fi}}%
3973   {\gdef\latinencoding{OT1}%
3974     \ifx\cf@encoding\bb@t@one
3975       \xdef\latinencoding{\bb@t@one}%
3976     \else
3977       \def\@elt#1{,#1,}%
3978       \edef\bb@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3979       \let\@elt\relax
3980       \bb@xin@{,T1,}\bb@tempa
3981       \ifin@
3982         \xdef\latinencoding{\bb@t@one}%
3983       \fi
3984     \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.

```

3985 \DeclareRobustCommand{\latintext}{%

```

```

3986 \fontencoding{\latinencoding}\selectfont
3987 \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.

```

3988 \ifx\undefined\DeclareTextFontCommand
3989 \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3990 \else
3991 \DeclareTextFontCommand{\textlatin}{\latintext}
3992 \fi

```

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

```

3993 \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.

```

3994 \bbl@trace{Loading basic (internal) bidi support}
3995 \ifodd\bbl@engine
3996 \else % TODO. Move to txtbabel. Any xe+lua bidi
3997 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3998 \bbl@error{bidi-only-lua}{\}\}\}%
3999 \let\bbl@beforeforeign\leavevmode
4000 \AtEndOfPackage{%
4001 \EnableBabelHook{babel-bidi}%
4002 \bbl@xebidipar}
4003 \fi\fi
4004 \def\bbl@loadxebidi#1{%
4005 \ifx\RTLfootnotetext\undefined
4006 \AtEndOfPackage{%
4007 \EnableBabelHook{babel-bidi}%
4008 \ifx\fontspec\undefined
4009 \usepackage{fontspec}% bidi needs fontspec
4010 \fi
4011 \usepackage#1{bidi}%
4012 \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
4013 \def\DigitsDotDashInterCharToks{% See the 'bidi' package
4014 \ifnum\@nameuse{bbl@wdir@\languagename}=\tw@ % 'AL' bidi
4015 \bbl@digitsdotdash % So ignore in 'R' bidi
4016 \fi}}%
4017 \fi}
4018 \ifnum\bbl@bidimode>200 % Any xe bidi=

```

```

4019 \ifcase\expandafter\@gobbletwo\the\bbbl@bidimode\or
4020 \bbbl@tentative{bidi=bidi}
4021 \bbbl@loadxebidi{
4022 \or
4023 \bbbl@loadxebidi{[rldocument]}
4024 \or
4025 \bbbl@loadxebidi{
4026 \fi
4027 \fi
4028 \fi
4029 % TODO? Separate:
4030 \ifnum\bbbl@bidimode=\@ne % bidi=default
4031 \let\bbbl@beforeforeign\leavevmode
4032 \ifodd\bbbl@engine % lua
4033 \newattribute\bbbl@attr@dir
4034 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbbl@attr@dir' }
4035 \bbbl@exp{\output{\bodydir\pagedir\the\output}}
4036 \fi
4037 \AtEndOfPackage{%
4038 \EnableBabelHook{babel-bidi}% pdf/lua/x
4039 \ifodd\bbbl@engine\else % pdf/x
4040 \bbbl@xebidipar
4041 \fi}
4042 \fi

```

Now come the macros used to set the direction when a language is switched. Testing are based on script names, because it's the user interface (including language and script in \babelprovide. First the (mostly) common macros.

```

4043 \bbbl@trace{Macros to switch the text direction}
4044 \def\bbbl@alscripts{,Arabic,Syriac,Thaana,}
4045 \def\bbbl@rscripts{%
4046 ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
4047 Old Hungarian,Lydian,Mandaean,Manichaeen,%
4048 Meroitic Cursive,Meroitic,Old North Arabian,%
4049 Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
4050 Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
4051 Old South Arabian,}%
4052 \def\bbbl@provide@dirs#1{%
4053 \bbbl@xin@{\csname bbl@sname@#1\endcsname}{\bbbl@alscripts\bbbl@rscripts}%
4054 \ifin@
4055 \global\bbbl@csarg\chardef{wdir@#1}\@ne
4056 \bbbl@xin@{\csname bbl@sname@#1\endcsname}{\bbbl@alscripts}%
4057 \ifin@
4058 \global\bbbl@csarg\chardef{wdir@#1}\tw@
4059 \fi
4060 \else
4061 \global\bbbl@csarg\chardef{wdir@#1}\z@
4062 \fi
4063 \ifodd\bbbl@engine
4064 \bbbl@csarg\ifcase{wdir@#1}%
4065 \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
4066 \or
4067 \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
4068 \or
4069 \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
4070 \fi
4071 \fi}
4072 \def\bbbl@switchdir{%
4073 \bbbl@ifunset{bbl@lsys@\languagename}{\bbbl@provide@lsys{\languagename}}{}%
4074 \bbbl@ifunset{bbl@wdir@\languagename}{\bbbl@provide@dirs{\languagename}}{}%
4075 \bbbl@exp{\bbbl@setdirs\bbbl@cl{wdir}}%
4076 \def\bbbl@setdirs#1{% TODO - math
4077 \ifcase\bbbl@select@type % TODO - strictly, not the right test

```



```

4078 \bbl@bodydir{#1}%
4079 \bbl@pardir{#1}% <- Must precede \bbl@textdir
4080 \fi
4081 \bbl@textdir{#1}}
4082 \ifnum\bbl@bidimode>\z@
4083 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4084 \DisableBabelHook{babel-bidi}
4085 \fi

```

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

```

4086 \ifodd\bbl@engine % luatex=1
4087 \else % pdftex=0, xetex=2
4088 \newcount\bbl@dirlevel
4089 \chardef\bbl@thetextdir\z@
4090 \chardef\bbl@thepardir\z@
4091 \def\bbl@textdir#1{%
4092 \ifcase#1\relax
4093 \chardef\bbl@thetextdir\z@
4094 \@nameuse{setlatin}%
4095 \bbl@textdir@i\beginL\endL
4096 \else
4097 \chardef\bbl@thetextdir\@ne
4098 \@nameuse{setnonlatin}%
4099 \bbl@textdir@i\beginR\endR
4100 \fi}
4101 \def\bbl@textdir@i#1#2{%
4102 \ifhmode
4103 \ifnum\currentgrouplevel>\z@
4104 \ifnum\currentgrouplevel=\bbl@dirlevel
4105 \bbl@error{multiple-bidi}{\}\}\}%
4106 \bgroup\aftergroup#2\aftergroup\egroup
4107 \else
4108 \ifcase\currentgroup\or % 0 bottom
4109 \aftergroup#2% 1 simple {}
4110 \or
4111 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4112 \or
4113 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4114 \or\or\or % vbox vtop align
4115 \or
4116 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4117 \or\or\or\or\or\or % output math disc insert vcent mathchoice
4118 \or
4119 \aftergroup#2% 14 \begingroup
4120 \else
4121 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4122 \fi
4123 \fi
4124 \bbl@dirlevel\currentgrouplevel
4125 \fi
4126 #1%
4127 \fi}
4128 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4129 \let\bbl@bodydir\@gobble
4130 \let\bbl@pagedir\@gobble
4131 \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).

```

4132 \def\bbl@xebidipar{%
4133 \let\bbl@xebidipar\relax
4134 \TeXeTstate\@ne
4135 \def\bbl@xeverypar{%

```

```

4136 \ifcase\bb@thepardir
4137 \ifcase\bb@thetextdir\else\beginR\fi
4138 \else
4139 {\setbox\z@\lastbox\beginR\box\z@}%
4140 \fi}%
4141 \AddToHook{para/begin}{\bb@xeverypar}}
4142 \ifnum\bb@bidimode>200 % Any xe bidi=
4143 \let\bb@textdir@i@gobbletwo
4144 \let\bb@xebidipar@empty
4145 \AddBabelHook{bidi}{foreign}{%
4146 \ifcase\bb@thetextdir
4147 \BabelWrapText{\LR{##1}}%
4148 \else
4149 \BabelWrapText{\RL{##1}}%
4150 \fi}
4151 \def\bb@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4152 \fi
4153 \fi

```

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

```

4154 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bb@textdir\z@#1}}
4155 \AtBeginDocument{%
4156 \ifx\pdfstringdefDisableCommands@undefined\else
4157 \ifx\pdfstringdefDisableCommands\relax\else
4158 \pdfstringdefDisableCommands{\let\babelsublr@firstofone}%
4159 \fi
4160 \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`.

```

4161 \bb@trace{Local Language Configuration}
4162 \ifx\loadlocalcfg@undefined
4163 \@ifpackagewith{babel}{noconfigs}%
4164 {\let\loadlocalcfg@gobble}%
4165 {\def\loadlocalcfg#1{%
4166 \InputIfFileExists{#1.cfg}%
4167 {\typeout{*****^J%
4168 * Local config file #1.cfg used^^J%
4169 *}}}%
4170 \@empty}}
4171 \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).

```

4172 \bb@trace{Language options}
4173 \let\bb@afterlang\relax
4174 \let\BabelModifiers\relax
4175 \let\bb@loaded@empty
4176 \def\bb@load@language#1{%
4177 \InputIfFileExists{#1.ldf}%
4178 {\edef\bb@loaded{\CurrentOption
4179 \ifx\bb@loaded@empty\else,\bb@loaded\fi}%
4180 \expandafter\let\expandafter\bb@afterlang
4181 \csname\CurrentOption.ldf-h@k\endcsname

```

```

4182 \expandafter\let\expandafter\BabelModifiers
4183 \csname bbl@mod@\CurrentOption\endcsname
4184 \bbl@expf{\AtBeginDocument}%
4185 \bbl@usehooks@lang{\CurrentOption}{beginndocument}{\CurrentOption}}}%
4186 {\IfFileExists{babel-#1.tex}%
4187 {\def\bbl@tempa{%
4188 .\There is a locale ini file for this language.\%
4189 If it's the main language, try adding `provide=*'\%
4190 to the babel package options}}%
4191 {\let\bbl@tempa\empty}%
4192 \bbl@error{unknown-package-option}{\}}}}

```

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.

```

4193 \def\bbl@try@load@lang#1#2#3{%
4194 \IfFileExists{\CurrentOption.ldf}%
4195 {\bbl@load@language{\CurrentOption}}%
4196 {#1\bbl@load@language{#2}#3}}
4197 %
4198 \DeclareOption{hebrew}{%
4199 \ifcase\bbl@engine\or
4200 \bbl@error{only-pdftex-lang}{hebrew}{\latex}}}%
4201 \fi
4202 \input{rlbabel.def}%
4203 \bbl@load@language{hebrew}}
4204 \DeclareOption{hungarian}{\bbl@try@load@lang}{magyar}}
4205 \DeclareOption{lowersorbian}{\bbl@try@load@lang}{lsorbian}}
4206 \DeclareOption{polutonikogreek}{%
4207 \bbl@try@load@lang}{greek}\languageattribute{greek}{polutoniko}}
4208 \DeclareOption{russian}{\bbl@try@load@lang}{russianb}}
4209 \DeclareOption{ukrainian}{\bbl@try@load@lang}{ukraineb}}
4210 \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.

```

4211 \ifx\bbl@opt@config@nnil
4212 \@ifpackagewith{babel}{noconfigs}}}%
4213 {\InputIfFileExists{bblopts.cfg}%
4214 {\typeout{*****^J%
4215 * Local config file bblopts.cfg used^^J%
4216 *}}}%
4217 {}}}%
4218 \else
4219 \InputIfFileExists{\bbl@opt@config.cfg}%
4220 {\typeout{*****^J%
4221 * Local config file \bbl@opt@config.cfg used^^J%
4222 *}}}%
4223 {\bbl@error{config-not-found}{\}}}}%
4224 \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.

```

4225 \ifx\bbl@opt@main@nnil
4226 \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4227 \let\bbl@tempb\empty
4228 \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4229 \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%

```

```

4230 \bbl@foreach\bbl@tempb{% \bbl@tempb is a reversed list
4231 \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4232 \ifodd\bbl@iniflag % = *=
4233 \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{%
4234 \else % n +=
4235 \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{%
4236 \fi
4237 \fi}%
4238 \fi
4239 \else
4240 \bbl@info{Main language set with 'main='. Except if you have\\%
4241 problems, prefer the default mechanism for setting\\%
4242 the main language, ie, as the last declared.\\%
4243 Reported}
4244 \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).

```

4245 \ifx\bbl@opt@main\@nnil\else
4246 \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
4247 \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4248 \fi

```

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

```

4249 \bbl@foreach\bbl@language@opts{%
4250 \def\bbl@tempa{#1}%
4251 \ifx\bbl@tempa\bbl@opt@main\else
4252 \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4253 \bbl@ifunset{ds@#1}%
4254 {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4255 {}%
4256 \else % + * (other = ini)
4257 \DeclareOption{#1}{%
4258 \bbl@ldfinit
4259 \babelprovide[import]{#1}%
4260 \bbl@afterldf{}}%
4261 \fi
4262 \fi}
4263 \bbl@foreach\@classoptionslist{%
4264 \def\bbl@tempa{#1}%
4265 \ifx\bbl@tempa\bbl@opt@main\else
4266 \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4267 \bbl@ifunset{ds@#1}%
4268 {\IfFileExists{#1.ldf}%
4269 {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4270 {}}%
4271 {}%
4272 \else % + * (other = ini)
4273 \IfFileExists{babel-#1.tex}%
4274 {\DeclareOption{#1}{%
4275 \bbl@ldfinit
4276 \babelprovide[import]{#1}%
4277 \bbl@afterldf{}}}%
4278 {}%
4279 \fi
4280 \fi}

```

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

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

```

4281 \def\AfterBabelLanguage#1{%
4282 \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang{}}

```

```
4283 \DeclareOption*{}
4284 \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.

```
4285 \bbl@trace{Option 'main'}
4286 \ifx\bbl@opt@main\@nnil
4287 \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4288 \let\bbl@tempc\@empty
4289 \edef\bbl@templ{,\bbl@loaded,}
4290 \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4291 \bbl@for\bbl@tempb\bbl@tempa{%
4292   \edef\bbl@tempd{,\bbl@tempb,}%
4293   \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4294   \bbl@xin@{\bbl@tempd}{\bbl@templ}%
4295   \ifin@{\edef\bbl@tempc{\bbl@tempb}}\fi}
4296 \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4297 \expandafter\bbl@tempa\bbl@loaded,\@nnil
4298 \ifx\bbl@tempb\bbl@tempc\else
4299   \bbl@warning{%
4300     Last declared language option is '\bbl@tempc',\%
4301     but the last processed one was '\bbl@tempb'.\%
4302     The main language can't be set as both a global\%
4303     and a package option. Use 'main=\bbl@tempc' as\%
4304     option. Reported}
4305 \fi
4306 \else
4307 \ifodd\bbl@iniflag % case 1,3 (main is ini)
4308   \bbl@ldfinit
4309   \let\CurrentOption\bbl@opt@main
4310   \bbl@exp{% \bbl@opt@provide = empty if *
4311     \\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4312   \bbl@afterldf{}
4313   \DeclareOption{\bbl@opt@main}{}
4314 \else % case 0,2 (main is ldf)
4315   \ifx\bbl@loadmain\relax
4316     \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4317   \else
4318     \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4319   \fi
4320   \ExecuteOptions{\bbl@opt@main}
4321   \@namedef{ds@\bbl@opt@main}{}%
4322 \fi
4323 \DeclareOption*{}
4324 \ProcessOptions*
4325 \fi
4326 \bbl@exp{%
4327   \\AtBeginDocument{\\bbl@usehooks@lang{/}{begindocument}{}}}%
4328 \def\AfterBabelLanguage{\bbl@error{late-after-babel}{}}{}
In order to catch the case where the user didn't specify a language we check whether
\bbl@main@language, has become defined. If not, the nil language is loaded.
4329 \ifx\bbl@main@language\@undefined
4330   \bbl@info{%
4331     You haven't specified a language as a class or package\%
4332     option. I'll load 'nil'. Reported}
4333   \bbl@load@language{nil}
4334 \fi
4335 </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

```
4336 (*kernel)
4337 \let\bbl@onlyswitch\@empty
4338 \input babel.def
4339 \let\bbl@onlyswitch\@undefined
4340 </kernel>
4341 %
4342 % \section{Error messages}
4343 %
4344 % They are loaded when |\bbl@error| is first called. To save space, the
4345 % main code just identifies them with a tag, and messages are stored in
4346 % a separate file. Since it can be loaded anywhere, you make sure some
4347 % catcodes have the right value, although those for |\, |', |^M|,
4348 % |%| and |=| are reset before loading the file.
4349 %
4350 (*errors)
4351 \catcode`\{=1 \catcode`\}=2 \catcode`\#=6
4352 \catcode`\:=12 \catcode`\,=12 \catcode`\.=12 \catcode`\- =12
4353 \catcode`\' =12 \catcode`\(=12 \catcode`\)=12
4354 \catcode`\@=11 \catcode`\^=7
4355 %
4356 \ifx\MessageBreak\@undefined
4357 \gdef\bbl@error@i#1#2{%
4358 \begingroup
4359 \newlinechar=`^^J
4360 \def\{^^J(babel) }%
4361 \errhelp{#2}\errmessage{\#1}%
4362 \endgroup}
4363 \else
4364 \gdef\bbl@error@i#1#2{%
4365 \begingroup
4366 \def\{MessageBreak}%
4367 \PackageError{babel}{#1}{#2}%
4368 \endgroup}
4369 \fi
4370 \def\bbl@errmessage#1#2#3{%
4371 \expandafter\gdef\cname bbl@err@#1\endcname##1##2##3{%
4372 \bbl@error@i{#2}{#3}}
4373 % Implicit #2#3#4:
4374 \gdef\bbl@error#1{\cname bbl@err@#1\endcname}
4375 %
4376 \bbl@errmessage{not-yet-available}
4377 {Not yet available}%
4378 {Find an armchair, sit down and wait}
4379 \bbl@errmessage{bad-package-option}%
4380 {Bad option '#1=#2'. Either you have misspelled the\\%
4381 key or there is a previous setting of '#1'. Valid\\%
4382 keys are, among others, 'shorthands', 'main', 'bidi',\\%
4383 'strings', 'config', 'headfoot', 'safe', 'math'.}%
4384 {See the manual for further details.}
4385 \bbl@errmessage{base-on-the-fly}
```

```

4386 {For a language to be defined on the fly 'base'\\%
4387 is not enough, and the whole package must be\\%
4388 loaded. Either delete the 'base' option or\\%
4389 request the languages explicitly}%
4390 {See the manual for further details.}
4391 \bbl@errmessage{undefined-language}
4392 {You haven't defined the language '#1' yet.\\%
4393 Perhaps you misspelled it or your installation\\%
4394 is not complete}%
4395 {Your command will be ignored, type <return> to proceed}
4396 \bbl@errmessage{shorthand-is-off}
4397 {I can't declare a shorthand turned off (\string#2)}
4398 {Sorry, but you can't use shorthands which have been\\%
4399 turned off in the package options}
4400 \bbl@errmessage{not-a-shorthand}
4401 {The character '\string #1' should be made a shorthand character;\\%
4402 add the command \string\useshorthands\string{#1\string} to
4403 the preamble.\\%
4404 I will ignore your instruction}%
4405 {You may proceed, but expect unexpected results}
4406 \bbl@errmessage{not-a-shorthand-b}
4407 {I can't switch '\string#2' on or off--not a shorthand}%
4408 {This character is not a shorthand. Maybe you made\\%
4409 a typing mistake? I will ignore your instruction.}
4410 \bbl@errmessage{unknown-attribute}
4411 {The attribute #2 is unknown for language #1.}%
4412 {Your command will be ignored, type <return> to proceed}
4413 \bbl@errmessage{missing-group}
4414 {Missing group for string \string#1}%
4415 {You must assign strings to some category, typically\\%
4416 captions or extras, but you set none}
4417 \bbl@errmessage{only-lua-xe}
4418 {This macro is available only in LuaLaTeX and XeLaTeX.}%
4419 {Consider switching to these engines.}
4420 \bbl@errmessage{only-lua}
4421 {This macro is available only in LuaLaTeX}%
4422 {Consider switching to that engine.}
4423 \bbl@errmessage{unknown-provide-key}
4424 {Unknown key '#1' in \string\babelprovide}%
4425 {See the manual for valid keys}%
4426 \bbl@errmessage{unknown-mapfont}
4427 {Option '\bbl@KVP@mapfont' unknown for\\%
4428 mapfont. Use 'direction'}%
4429 {See the manual for details.}
4430 \bbl@errmessage{no-ini-file}
4431 {There is no ini file for the requested language\\%
4432 (#1: \languagename). Perhaps you misspelled it or your\\%
4433 installation is not complete}%
4434 {Fix the name or reinstall babel.}
4435 \bbl@errmessage{digits-is-reserved}
4436 {The counter name 'digits' is reserved for mapping\\%
4437 decimal digits}%
4438 {Use another name.}
4439 \bbl@errmessage{limit-two-digits}
4440 {Currently two-digit years are restricted to the\\%
4441 range 0-9999}%
4442 {There is little you can do. Sorry.}
4443 \bbl@errmessage{alphabetic-too-large}
4444 {Alphabetic numeral too large (#1)}%
4445 {Currently this is the limit.}
4446 \bbl@errmessage{no-ini-info}
4447 {I've found no info for the current locale.\\%
4448 The corresponding ini file has not been loaded\\%

```

```

4449     Perhaps it doesn't exist}%
4450     {See the manual for details.}
4451 \bbl@errmessage{unknown-ini-field}
4452     {Unknown field '#1' in \string\BCPdata.\%
4453     Perhaps you misspelled it}%
4454     {See the manual for details.}
4455 \bbl@errmessage{unknown-locale-key}
4456     {Unknown key for locale '#2':\%
4457     #3\%
4458     \string#1 will be set to \string\relax}%
4459     {Perhaps you misspelled it.}%
4460 \bbl@errmessage{adjust-only-vertical}
4461     {Currently, #1 related features can be adjusted only\%
4462     in the main vertical list}%
4463     {Maybe things change in the future, but this is what it is.}
4464 \bbl@errmessage{layout-only-vertical}
4465     {Currently, layout related features can be adjusted only\%
4466     in vertical mode}%
4467     {Maybe things change in the future, but this is what it is.}
4468 \bbl@errmessage{bidi-only-lua}
4469     {The bidi method 'basic' is available only in\%
4470     luatex. I'll continue with 'bidi=default', so\%
4471     expect wrong results}%
4472     {See the manual for further details.}
4473 \bbl@errmessage{multiple-bidi}
4474     {Multiple bidi settings inside a group}%
4475     {I'll insert a new group, but expect wrong results.}
4476 \bbl@errmessage{unknown-package-option}
4477     {Unknown option '\CurrentOption'. Either you misspelled it\%
4478     or the language definition file \CurrentOption.ldf\%
4479     was not found%
4480     \bbl@tempa}
4481     {Valid options are, among others: shorthands=, KeepShorthandsActive,\%
4482     activeacute, activegrave, noconfigs, safe=, main=, math=\%
4483     headfoot=, strings=, config=, hyphenmap=, or a language name.}
4484 \bbl@errmessage{config-not-found}
4485     {Local config file '\bbl@opt@config.cfg' not found}%
4486     {Perhaps you misspelled it.}
4487 \bbl@errmessage{late-after-babel}
4488     {Too late for \string\AfterBabelLanguage}%
4489     {Languages have been loaded, so I can do nothing}
4490 \bbl@errmessage{double-hyphens-class}
4491     {Double hyphens aren't allowed in \string\babelcharclass\%
4492     because it's potentially ambiguous}%
4493     {See the manual for further info}
4494 \bbl@errmessage{unknown-interchar}
4495     {'#1' for '\languagename' cannot be enabled.\%
4496     Maybe there is a typo}%
4497     {See the manual for further details.}
4498 \bbl@errmessage{unknown-interchar-b}
4499     {'#1' for '\languagename' cannot be disabled.\%
4500     Maybe there is a typo}%
4501     {See the manual for further details.}
4502 \bbl@errmessage{charproperty-only-vertical}
4503     {\string\babelcharproperty\space can be used only in\%
4504     vertical mode (preamble or between paragraphs)}%
4505     {See the manual for further info}
4506 \bbl@errmessage{unknown-char-property}
4507     {No property named '#2'. Allowed values are\%
4508     direction (bc), mirror (bmg), and linebreak (lb)}%
4509     {See the manual for further info}
4510 \bbl@errmessage{bad-transform-option}
4511     {Bad option '#1' in a transform.\%

```



```

4512 I'll ignore it but expect more errors}%
4513 {See the manual for further info.}
4514 \bbl@errmessage{font-conflict-transforms}
4515 {Transforms cannot be re-assigned to different\\%
4516 fonts. The conflict is in '\bbl@kv@label'.\\%
4517 Apply the same fonts or use a different label}%
4518 {See the manual for further details.}
4519 \bbl@errmessage{transform-not-available}
4520 {'#1' for '\languagename' cannot be enabled.\\%
4521 Maybe there is a typo or it's a font-dependent transform}%
4522 {See the manual for further details.}
4523 \bbl@errmessage{transform-not-available-b}
4524 {'#1' for '\languagename' cannot be disabled.\\%
4525 Maybe there is a typo or it's a font-dependent transform}%
4526 {See the manual for further details.}
4527 \bbl@errmessage{year-out-range}
4528 {Year out of range.\\%
4529 The allowed range is #1}%
4530 {See the manual for further details.}
4531 \bbl@errmessage{only-pdftex-lang}
4532 {The '#1' ldf style doesn't work with #2,\\%
4533 but you can use the ini locale instead.\\%
4534 Try adding 'provide=*' to the option list. You may\\%
4535 also want to set 'bidi=' to some value}%
4536 {See the manual for further details.}
4537 \bbl@errmessage{hyphenmins-args}
4538 {\string\babelhyphenmins\ accepts either the optional\\%
4539 argument or the star, but not both at the same time}%
4540 {See the manual for further details.}
4541 </errors>
4542 <*patterns>

```

7 Loading hyphenation patterns

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

```

4543 <@Make sure ProvidesFile is defined@>
4544 \ProvidesFile{hyphen.cfg}[<@date@> v<@version@> Babel hyphens]
4545 \xdef\bbl@format{\jobname}
4546 \def\bbl@version{<@version@>}
4547 \def\bbl@date{<@date@>}
4548 \ifx\AtBeginDocument\@undefined
4549 \def\@empty{}
4550 \fi
4551 <@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.

```

4552 \def\process@line#1#2 #3 #4 {%
4553 \ifx=#1%
4554 \process@synonym{#2}%
4555 \else
4556 \process@language{#1#2}{#3}{#4}%
4557 \fi
4558 \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.

```

4559 \toks@{}
4560 \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 `hyphenmins` parameters for the synonym.

```

4561 \def\process@synonym#1{%
4562   \ifnum\last@language=\m@ne
4563     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4564   \else
4565     \expandafter\chardef\csname l@#1\endcsname\last@language
4566     \wlog{\string\l@#1=\string\language\the\last@language}%
4567     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4568       \csname\language\hyphenmins\endcsname
4569     \let\bb@elt\relax
4570     \edef\bb@languages{\bb@languages\bb@elt{#1}{\the\last@language}}}%
4571   \fi}

```

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

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

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

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

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

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

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

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

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

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

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

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

```

4572 \def\process@language#1#2#3{%
4573   \expandafter\addlanguage\csname l@#1\endcsname
4574   \expandafter\language\csname l@#1\endcsname
4575   \edef\language#1%
4576   \bb@hook@everylanguage{#1}%
4577   % > luatex
4578   \bb@get@enc#1: :@@@
4579   \begingroup
4580     \lefthyphenmin\m@ne
4581     \bb@hook@loadpatterns{#2}%
4582     % > luatex
4583     \ifnum\lefthyphenmin=\m@ne
4584     \else
4585       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4586         \the\lefthyphenmin\the\righthyphenmin}%
4587     \fi
4588   \endgroup
4589   \def\bb@tempa{#3}%
4590   \ifx\bb@tempa@empty\else

```

```

4591 \bbl@hook@loadexceptions{#3}%
4592 % > luatex
4593 \fi
4594 \let\bbl@elt\relax
4595 \edef\bbl@languages{%
4596 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4597 \ifnum\the\language=\z@
4598 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4599 \set@hyphenmins\tw@\thr@@\relax
4600 \else
4601 \expandafter\expandafter\expandafter\set@hyphenmins
4602 \csname #1hyphenmins\endcsname
4603 \fi
4604 \the\toks@
4605 \toks@{}%
4606 \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.

```

4607 \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.

```

4608 \def\bbl@hook@everylanguage#1{}
4609 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4610 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4611 \def\bbl@hook@loadkernel#1{%
4612 \def\addlanguage{\csname newlanguage\endcsname}%
4613 \def\adddialect##1##2{%
4614 \global\chardef##1##2\relax
4615 \wlog{\string##1 = a dialect from \string\language##2}}%
4616 \def\iflanguage##1{%
4617 \expandafter\ifx\csname l@##1\endcsname\relax
4618 \@noLanerr{##1}%
4619 \else
4620 \ifnum\csname l@##1\endcsname=\language
4621 \expandafter\expandafter\expandafter\@firstoftwo
4622 \else
4623 \expandafter\expandafter\expandafter\@secondoftwo
4624 \fi
4625 \fi}%
4626 \def\providehyphenmins##1##2{%
4627 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4628 \@namedef{##1hyphenmins}{##2}%
4629 \fi}%
4630 \def\set@hyphenmins##1##2{%
4631 \leftthyphenmin##1\relax
4632 \rightthyphenmin##2\relax}%
4633 \def\selectlanguage{%
4634 \errhelp{Selecting a language requires a package supporting it}%
4635 \errmessage{Not loaded}}%
4636 \let\foreignlanguage\selectlanguage
4637 \let\otherlanguage\selectlanguage
4638 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4639 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4640 \def\setlocale{%
4641 \errhelp{Find an armchair, sit down and wait}%
4642 \errmessage{(babel) Not yet available}}%
4643 \let\uselocale\setlocale
4644 \let\locale\setlocale
4645 \let\selectlocale\setlocale
4646 \let\localename\setlocale

```

```

4647 \let\textlocale\setlocale
4648 \let\textlanguage\setlocale
4649 \let\language\text\setlocale}
4650 \begingroup
4651 \def\AddBabelHook#1#2{%
4652   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4653     \def\next{\toks1}%
4654   \else
4655     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4656   \fi
4657   \next}
4658 \if\directlua@undefined
4659   \if\XeTeXinputencoding@undefined\else
4660     \input xebabel.def
4661   \fi
4662 \else
4663   \input luababel.def
4664 \fi
4665 \openin1 = babel-\bbl@format.cfg
4666 \ifeof1
4667 \else
4668   \input babel-\bbl@format.cfg\relax
4669 \fi
4670 \closein1
4671 \endgroup
4672 \bbl@hook@loadkernel{switch.def}

```

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

```
4673 \openin1 = language.dat
```

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

```

4674 \def\language\name{english}%
4675 \ifeof1
4676   \message{I couldn't find the file language.dat,\space
4677             I will try the file hyphen.tex}
4678   \input hyphen.tex\relax
4679   \chardef\l@english\z@
4680 \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`.

```
4681 \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.

```

4682 \loop
4683   \endlinechar\m@ne
4684   \read1 to \bbl@line
4685   \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.

```

4686   \if T\ifeof1\fi T\relax
4687   \ifx\bbl@line@empty\else
4688     \edef\bbl@line{\bbl@line\space\space\space}%
4689     \expandafter\process@line\bbl@line\relax
4690   \fi
4691 \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.

```
4692 \begingroup
4693   \def\bbl@elt#1#2#3#4{%
4694     \global\language=#2\relax
4695     \gdef\languagename{#1}%
4696     \def\bbl@elt##1##2##3##4{}}%
4697   \bbl@languages
4698 \endgroup
4699 \fi
4700 \closeinl
```

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

```
4701 \if/\the\toks@/\else
4702   \errhelp{language.dat loads no language, only synonyms}
4703   \errmessage{Orphan language synonym}
4704 \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.

```
4705 \let\bbl@line@\undefined
4706 \let\process@line@\undefined
4707 \let\process@synonym@\undefined
4708 \let\process@language@\undefined
4709 \let\bbl@get@enc@\undefined
4710 \let\bbl@hyph@enc@\undefined
4711 \let\bbl@tempa@\undefined
4712 \let\bbl@hook@loadkernel@\undefined
4713 \let\bbl@hook@everylanguage@\undefined
4714 \let\bbl@hook@loadpatterns@\undefined
4715 \let\bbl@hook@loadexceptions@\undefined
4716 </patterns>
```

Here the code for `iniTEX` ends.

8 Font handling with fontspec

Add the bidi handler just before `luaoftload`, 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].

```
4717 <<{*More package options}>> ≡
4718 \chardef\bbl@bidimode\z@
4719 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4720 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4721 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4722 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4723 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4724 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4725 <</More package options>>
```

With explicit languages, we could define the font at once, but we don't. Just wait and see if the language is actually activated. `bbl@font` replaces hardcoded font names inside `\. . family` by the corresponding macro `\. . default`.

```
4726 <<{*Font selection}>> ≡
4727 \bbl@trace{Font handling with fontspec}
4728 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4729 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ccheckstdfonts}
4730 \DisableBabelHook{babel-fontspec}
4731 \@onlypreamble\babelfont
4732 \newcommand\babelfont[2][]{% 1=langs/scripts 2=fam
4733   \bbl@foreach{#1}{%
4734     \expandafter\ifx\csname date##1\endcsname\relax
```

```

4735 \IfFileExists{babel-##1.tex}%
4736 {\babelprovide{##1}}%
4737 }%
4738 \fi}%
4739 \edef\bbbl@tempa{#1}%
4740 \def\bbbl@tempb{#2}% Used by \bbbl@bblfont
4741 \ifx\fontspec\@undefined
4742 \usepackage{fontspec}%
4743 \fi
4744 \EnableBabelHook{babel-fontspec}%
4745 \bbbl@bblfont}
4746 \newcommand\bbbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4747 \bbbl@ifunset{\bbbl@tempb family}%
4748 {\bbbl@providfam{\bbbl@tempb}}%
4749 }%
4750 % For the default font, just in case:
4751 \bbbl@ifunset{\bbbl@lsys@\languagename}{\bbbl@provide@lsys{\languagename}}{}%
4752 \expandafter\bbbl@ifblank\expandafter{\bbbl@tempa}%
4753 {\bbbl@csarg\edef{\bbbl@tempb dflt@}{<{#1}{#2}}% save \bbbl@rmdflt@
4754 \bbbl@exp{%
4755 \let\<\bbbl@tempb dflt@\languagename>\<\bbbl@tempb dflt@>%
4756 \\\bbbl@font@set\<\bbbl@tempb dflt@\languagename>%
4757 \<\bbbl@tempb default>\<\bbbl@tempb family>}}%
4758 {\bbbl@foreach\bbbl@tempa{% ie \bbbl@rmdflt@lang / *scrt
4759 \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:

```

4760 \def\bbbl@providfam#1{%
4761 \bbbl@exp{%
4762 \\\newcommand\<#1default>{}% Just define it
4763 \\\bbbl@add@list\\bbbl@font@fams{#1}%
4764 \\\DeclareRobustCommand\<#1family>{%
4765 \\\not@math@alphabet\<#1family>\relax
4766 % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4767 \\\fontfamily\<#1default>%
4768 \<ifx>\\\UseHooks\\@undefined\<else>\\\UseHook{#1family}\<fi>%
4769 \\\selectfont}%
4770 \\\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.

```

4771 \def\bbbl@nostdfont#1{%
4772 \bbbl@ifunset{\bbbl@WFF@\f@family}%
4773 {\bbbl@csarg\gdef{WFF@\f@family}}{}% Flag, to avoid dupl warns
4774 \bbbl@infowarn{The current font is not a babel standard family:\%
4775 #1%
4776 \fontname\font\\%
4777 There is nothing intrinsically wrong with this warning, and\\%
4778 you can ignore it altogether if you do not need these\\%
4779 families. But if they are used in the document, you should be\\%
4780 aware 'babel' will not set Script and Language for them, so\\%
4781 you may consider defining a new family with \string\babelfont.\\%
4782 See the manual for further details about \string\babelfont.\\%
4783 Reported}}
4784 }{}%
4785 \gdef\bbbl@switchfont{%
4786 \bbbl@ifunset{\bbbl@lsys@\languagename}{\bbbl@provide@lsys{\languagename}}{}%
4787 \bbbl@exp{% eg Arabic -> arabic
4788 \lowercase{\edef\\bbbl@tempa{\bbbl@c{l}{sname}}}}%
4789 \bbbl@foreach\bbbl@font@fams{%
4790 \bbbl@ifunset{\bbbl@##1dflt@\languagename}% (1) language?
4791 {\bbbl@ifunset{\bbbl@##1dflt@*\bbbl@tempa}% (2) from script?
4792 {\bbbl@ifunset{\bbbl@##1dflt@}% 2=F - (3) from generic?
4793 }{}% 123=F - nothing!

```

```

4794      {\bbl@exp{%          3=T - from generic
4795      \global\let\<bbl@##1dflt@\language>%
4796      \<bbl@##1dflt@>}}}%
4797      {\bbl@exp{%          2=T - from script
4798      \global\let\<bbl@##1dflt@\language>%
4799      \<bbl@##1dflt@*\bbl@tempa>}}}%
4800      {}}}%          1=T - language, already defined
4801 \def\bbl@tempa{\bbl@nostdfont{}}% TODO. Don't use \bbl@tempa
4802 \bbl@foreach\bbl@font@fams{% don't gather with prev for
4803 \bbl@ifunset{bbl@##1dflt@\language}%
4804 {\bbl@cs{famrst@##1}%
4805 \global\bbl@csarg\let{famrst@##1}\relax}%
4806 {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4807 \\\bbl@add\\\originalTeX%
4808 \\\bbl@font@rst{\bbl@c\{##1dflt}}%
4809 \<##1default>\<##1family>{##1}}}%
4810 \\\bbl@font@set\<bbl@##1dflt@\language>% the main part!
4811 \<##1default>\<##1family>}}}%
4812 \bbl@ifrestoring{ }\bbl@tempa}}%

```

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

```

4813 \ifx\f@family\undefined\else % if latex
4814 \ifcase\bbl@engine % if pdftex
4815 \let\bbl@cckcckstdfonts\relax
4816 \else
4817 \def\bbl@cckcckstdfonts{%
4818 \begingroup
4819 \global\let\bbl@cckcckstdfonts\relax
4820 \let\bbl@tempa\empty
4821 \bbl@foreach\bbl@font@fams{%
4822 \bbl@ifunset{bbl@##1dflt@}%
4823 {\@nameuse{##1family}%
4824 \bbl@csarg\gdef{WFF@f@family}}}% Flag
4825 \bbl@exp{\\\bbl@add\\\bbl@tempa{* \<##1family>= \f@family\\}%
4826 \space\space\fontname\font\\}}}%
4827 \bbl@csarg\xdef{##1dflt@}{\f@family}%
4828 \expandafter\xdef{csname ##1default\endcsname{\f@family}}%
4829 {}}}%
4830 \ifx\bbl@tempa\empty\else
4831 \bbl@infowarn{The following font families will use the default\\%
4832 settings for all or some languages:\\%
4833 \bbl@tempa
4834 There is nothing intrinsically wrong with it, but\\%
4835 'babel' will no set Script and Language, which could\\%
4836 be relevant in some languages. If your document uses\\%
4837 these families, consider redefining them with \string\babelfont.\\%
4838 Reported}%
4839 \fi
4840 \endgroup}
4841 \fi
4842 \fi

```

Now the macros defining the font with `fontspec`.

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

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

```

4843 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily

```

```

4844 \bbl@xin@{<>}{#1}%
4845 \ifin@
4846 \bbl@exp{\\bbl@fontspec@set\\#1\expandafter@gobbletwo#1\\#3}%
4847 \fi
4848 \bbl@exp{% 'Unprotected' macros return prev values
4849 \def\\#2{#1}% eg, \rmdefault{\bbl@rmdflt@lang}
4850 \\bbl@ifsamestring{#2}{\f@family}%
4851 {\#3%
4852 \\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}}}%
4853 \let\\bbl@tempa\relax}%
4854 {}}}
4855 % TODO - next should be global?, but even local does its job. I'm
4856 % still not sure -- must investigate:
4857 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4858 \let\bbl@tempe\bbl@mapselect
4859 \edef\bbl@tempb{\bbl@stripslash#4/}% Catcodes hack (better pass it).
4860 \bbl@exp{\\bbl@replace\\bbl@tempb{\bbl@stripslash\family/}}}%
4861 \let\bbl@mapselect\relax
4862 \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4863 \let#4\@empty % Make sure \renewfontfamily is valid
4864 \bbl@exp{%
4865 \let\\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
4866 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4867 {\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4868 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4869 {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4870 \\renewfontfamily\\#4%
4871 [\bbl@cl{lsys},% xetex removes unknown features :- (
4872 \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
4873 #2]}{#3}% ie \bbl@exp{..}{#3}
4874 \begingroup
4875 #4%
4876 \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4877 \endgroup % TODO. Find better tests:
4878 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4879 {\expandafter\meaning\cscname TU/#1/bx/sc\endcsname}%
4880 \ifin@
4881 \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
4882 \fi
4883 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4884 {\expandafter\meaning\cscname TU/#1/bx/scit\endcsname}%
4885 \ifin@
4886 \global\bbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
4887 \fi
4888 \let#4\bbl@temp@fam
4889 \bbl@exp{\let<\bbl@stripslash#4\space>}\bbl@temp@pfam
4890 \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.

```

4891 \def\bbl@font@rst#1#2#3#4{%
4892 \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`.

```

4893 \def\bbl@font@fams{rm,sf,tt}
4894 <</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.


```

4895 <<(*Footnote changes)>> ≡
4896 \bbl@trace{Bidi footnotes}
4897 \ifnum\bbl@bidimode>\z@ % Any bidi=
4898 \def\bbl@footnote#1#2#3{%
4899   \@ifnextchar[%
4900     {\bbl@footnote@o{#1}{#2}{#3}}%
4901     {\bbl@footnote@x{#1}{#2}{#3}}}
4902 \long\def\bbl@footnote@x#1#2#3#4{%
4903   \bgroup
4904     \select@language@x{\bbl@main@language}%
4905     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4906   \egroup}
4907 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4908   \bgroup
4909     \select@language@x{\bbl@main@language}%
4910     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4911   \egroup}
4912 \def\bbl@footnotetext#1#2#3{%
4913   \@ifnextchar[%
4914     {\bbl@footnotetext@o{#1}{#2}{#3}}%
4915     {\bbl@footnotetext@x{#1}{#2}{#3}}}
4916 \long\def\bbl@footnotetext@x#1#2#3#4{%
4917   \bgroup
4918     \select@language@x{\bbl@main@language}%
4919     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4920   \egroup}
4921 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4922   \bgroup
4923     \select@language@x{\bbl@main@language}%
4924     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4925   \egroup}
4926 \def\BabelFootnote#1#2#3#4{%
4927   \ifx\bbl@fn@footnote\@undefined
4928     \let\bbl@fn@footnote\footnote
4929   \fi
4930   \ifx\bbl@fn@footnotetext\@undefined
4931     \let\bbl@fn@footnotetext\footnotetext
4932   \fi
4933   \bbl@ifblank{#2}%
4934     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4935     \@namedef{\bbl@stripslash#1text}%
4936       {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4937     {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4938     \@namedef{\bbl@stripslash#1text}%
4939       {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}}
4940 \fi
4941 <</Footnote changes>>

```

Now, the code.

```

4942 (*xetex)
4943 \def\BabelStringsDefault{unicode}
4944 \let\xebbl@stop\relax
4945 \AddBabelHook{xetex}{encodedcommands}{%
4946   \def\bbl@tempa{#1}%
4947   \ifx\bbl@tempa\@empty
4948     \XeTeXinputencoding"bytes"%
4949   \else
4950     \XeTeXinputencoding"#1"%
4951   \fi
4952   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4953 \AddBabelHook{xetex}{stopcommands}{%
4954   \xebbl@stop
4955   \let\xebbl@stop\relax}

```

```

4956 \def\bb@input@classes{% Used in CJK intraspaces
4957 \input{load-unicode-xetex-classes.tex}%
4958 \let\bb@input@classes\relax}
4959 \def\bb@intraspace#1 #2 #3\@@{%
4960 \bb@csarg\gdef{xeisp@\languagename}%
4961 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4962 \def\bb@intrapenalty#1\@@{%
4963 \bb@csarg\gdef{xeipn@\languagename}%
4964 {\XeTeXlinebreakpenalty #1\relax}}
4965 \def\bb@provide@intraspace{%
4966 \bb@xin@{/s}{/\bb@cl{lnbrk}}%
4967 \ifin@else\bb@xin@{/c}{/\bb@cl{lnbrk}}\fi
4968 \ifin@
4969 \bb@ifunset{bb@intsp@\languagename}{}%
4970 {\expandafter\ifx\csname bb@intsp@\languagename\endcsname\@empty\else
4971 \ifx\bb@KVP@intraspace\@nnil
4972 \bb@exp{%
4973 \\\bb@intraspace\bb@cl{intsp}\@@}%
4974 \fi
4975 \ifx\bb@KVP@intrapenalty\@nnil
4976 \bb@intrapenalty0\@@
4977 \fi
4978 \fi
4979 \ifx\bb@KVP@intraspace\@nnil\else % We may override the ini
4980 \expandafter\bb@intraspace\bb@KVP@intraspace\@@
4981 \fi
4982 \ifx\bb@KVP@intrapenalty\@nnil\else
4983 \expandafter\bb@intrapenalty\bb@KVP@intrapenalty\@@
4984 \fi
4985 \bb@exp{%
4986 % TODO. Execute only once (but redundant):
4987 \\\bb@add\<extras\languagename>{%
4988 \XeTeXlinebreaklocale "\bb@cl{tbc}"%
4989 \<bb@xeisp@\languagename>%
4990 \<bb@xeipn@\languagename>}%
4991 \\\bb@tglobal\<extras\languagename>%
4992 \\\bb@add\<noextras\languagename>{%
4993 \XeTeXlinebreaklocale ""}%
4994 \\\bb@tglobal\<noextras\languagename>}%
4995 \ifx\bb@ispacesize\@undefined
4996 \gdef\bb@ispacesize{\bb@cl{xeisp}}%
4997 \ifx\AtBeginDocument\@notprerr
4998 \expandafter\@secondoftwo % to execute right now
4999 \fi
5000 \AtBeginDocument{\bb@patchfont{\bb@ispacesize}}%
5001 \fi}%
5002 \fi}
5003 \ifx\DisableBabelHook\@undefined\endinput\fi %%% TODO: why
5004 <@Font selection@>
5005 \def\bb@provide@extra#1{}

```

10 Support for interchar

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

```

5006 \ifnum\Xe@alloc@intercharclass<\thr@
5007 \Xe@alloc@intercharclass\thr@
5008 \fi
5009 \chardef\bb@xe@class@default@=\z@
5010 \chardef\bb@xe@class@cjkideogram@=\@ne
5011 \chardef\bb@xe@class@cjkleftpunctuation@=\tw@
5012 \chardef\bb@xe@class@cjkrightpunctuation@=\thr@

```

```
5013 \chardef\bblexeclass@boundary@=4095
5014 \chardef\bblexeclass@ignore@=4096
```

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

```
5015 \AddBabelHook{babel-interchar}{beforeextras}{%
5016   \@nameuse{bblexechars@\languagename}}
5017 \DisableBabelHook{babel-interchar}
5018 \protected\def\bb@charclass#1{%
5019   \ifnum\count@<\z@
5020     \count@-\count@
5021     \loop
5022       \bb@exp{%
5023         \\babel@savevariable{\XeTeXcharclass`Uchar\count@}}%
5024         \XeTeXcharclass\count@ \bb@tempc
5025         \ifnum\count@<`#1\relax
5026           \advance\count@\@ne
5027         \repeat
5028   \else
5029     \babel@savevariable{\XeTeXcharclass`#1}%
5030     \XeTeXcharclass`#1 \bb@tempc
5031   \fi
5032   \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 `\bb@usingxeclass\bblexeclass@punct@english\bb@charclass{.}` `\bb@charclass{,}` (etc.), where `\bb@usingxeclass` stores the class to be applied to the subsequent characters. The `\ifcat` part deals with the alternative way to enter characters as macros (eg, `\`). As a special case, hyphens are stored as `\bb@upto`, to deal with ranges.

```
5033 \newcommand\bb@ifinterchar[1]{%
5034   \let\bb@tempa@gobble      % Assume to ignore
5035   \edef\bb@tempb{\zap@space#1 \@empty}%
5036   \ifx\bb@KVP@interchar\@nnil\else
5037     \bb@replace\bb@KVP@interchar{ }{,}%
5038     \bb@foreach\bb@tempb{%
5039       \bb@xin@{,##1,}{, \bb@KVP@interchar,}%
5040     \ifin@
5041       \let\bb@tempa\@firstofone
5042     \fi}%
5043   \fi
5044   \bb@tempa}
5045 \newcommand\IfBabelIntercharT[2]{%
5046   \bb@carg\bb@add{\bb@icsave@CurrentOption}{\bb@ifinterchar{#1}{#2}}%
5047 \newcommand\babelcharclass[3]{%
5048   \EnableBabelHook{babel-interchar}%
5049   \bb@csarg\newXeTeXintercharclass{xeclass@#2@#1}%
5050   \def\bb@tempb##1{%
5051     \ifx##1@empty\else
5052       \ifx##1-%
5053         \bb@upto
5054       \else
5055         \bb@charclass%
5056         \ifcat\noexpand##1\relax\bb@stripslash##1\else\string##1\fi}%
5057     \fi
5058     \expandafter\bb@tempb
5059   \fi}%
5060 \bb@ifunset{bblexechars@#1}%
5061   {\toks@%
5062     \babel@savevariable\XeTeXinterchartokenstate
5063     \XeTeXinterchartokenstate\@ne
5064   }%}
```

```

5065     {\toks@\expandafter\expandafter\expandafter{%
5066         \csname bbl@xechars@#1\endcsname}}}%
5067 \bbl@csarg\edef{xechars@#1}{%
5068     \the\toks@
5069     \bbl@usingxeclasse\csname bbl@xeclasse@#2@#1\endcsname
5070     \bbl@tempb#3\@empty}}
5071 \protected\def\bbl@usingxeclasse#1{\count@\z@ \let\bbl@tempc#1}
5072 \protected\def\bbl@upto{%
5073     \ifnum\count@>\z@
5074         \advance\count@\@ne
5075         \count@-\count@
5076     \else\ifnum\count@=\z@
5077         \bbl@charclass{-}%
5078     \else
5079         \bbl@error{double-hyphens-class}{\}\}\}%
5080     \fi\fi}

```

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

```

5081 \def\bbl@ignoreinterchar{%
5082     \ifnum\language=\@nohyphenation
5083         \expandafter\@gobble
5084     \else
5085         \expandafter\@firstofone
5086     \fi}
5087 \newcommand\babelinterchar[5][{}]{%
5088     \let\bbl@kv@label\@empty
5089     \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}}%
5090     \@namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
5091     {\bbl@ignoreinterchar{#5}}%
5092     \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone
5093     \bbl@exp{\bbl@for{\bbl@tempa{\zap@space#3 \@empty}}}%
5094     \bbl@exp{\bbl@for{\bbl@tempb{\zap@space#4 \@empty}}}%
5095     \XeTeXinterchartoks
5096         \@nameuse{bbl@xeclasse@\bbl@tempa @%
5097             \bbl@ifunset{bbl@xeclasse@\bbl@tempa @#2}{\}\}\} %
5098         \@nameuse{bbl@xeclasse@\bbl@tempb @%
5099             \bbl@ifunset{bbl@xeclasse@\bbl@tempb @#2}{\}\}\} %
5100     = \expandafter{%
5101         \csname bbl@ic@\bbl@kv@label @#2\expandafter\endcsname
5102         \csname\zap@space bbl@xeinter@\bbl@kv@label
5103             @#3@#4@#2 \@empty\endcsname}}}}
5104 \DeclareRobustCommand\enablelocaleinterchar[1]{%
5105     \bbl@ifunset{bbl@ic@#1@languagename}%
5106     {\bbl@error{unknown-interchar}{#1}{\}\}\}%
5107     {\bbl@csarg\let{ic@#1@languagename}\@firstofone}}
5108 \DeclareRobustCommand\disablelocaleinterchar[1]{%
5109     \bbl@ifunset{bbl@ic@#1@languagename}%
5110     {\bbl@error{unknown-interchar-b}{#1}{\}\}\}%
5111     {\bbl@csarg\let{ic@#1@languagename}\@gobble}}
5112 </xetex>

```

10.1 Layout

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

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

Consider `txbtabel` as a shorthand for `tex-xet babel`, which is the bidi model in both `pdftex` and `xetex`.

```

5113 < *xetex | texxet >
5114 \providecommand\bbl@provide@intraspace{}

```

```

5115 \bbl@trace{Redefinitions for bidi layout}
5116 \def\bbl@sspre@caption{% TODO: Unused!
5117   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
5118 \ifx\bbl@opt@layout\@nnil\else % if layout=..
5119 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
5120 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
5121 \ifnum\bbl@bidimode>\z@ % TODO: always?
5122   \def\@hangfrom#1{%
5123     \setbox\@tempboxa\hbox{#1}%
5124     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
5125     \noindent\box\@tempboxa}
5126   \def\raggedright{%
5127     \let\@centercr
5128     \bbl@startskip\z@skip
5129     \@rightskip\@flushglue
5130     \bbl@endskip\@rightskip
5131     \parindent\z@
5132     \parfillskip\bbl@startskip}
5133   \def\raggedleft{%
5134     \let\@centercr
5135     \bbl@startskip\@flushglue
5136     \bbl@endskip\z@skip
5137     \parindent\z@
5138     \parfillskip\bbl@endskip}
5139 \fi
5140 \IfBabelLayout{lists}
5141   {\bbl@sreplace\list
5142     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
5143     \def\bbl@listleftmargin{%
5144       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
5145     \ifcase\bbl@engine
5146       \def\labelenumii{\theenumii}% pdftex doesn't reverse ()
5147       \def\p@enumiii{\p@enumii}\theenumii}%
5148     \fi
5149     \bbl@sreplace\@verbatim
5150       {\leftskip\@totalleftmargin}%
5151       {\bbl@startskip\textwidth
5152         \advance\bbl@startskip-\linewidth}%
5153     \bbl@sreplace\@verbatim
5154       {\rightskip\z@skip}%
5155       {\bbl@endskip\z@skip}}%
5156   {}
5157 \IfBabelLayout{contents}
5158   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
5159     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
5160   {}
5161 \IfBabelLayout{columns}
5162   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
5163     \def\bbl@outputbox#1{%
5164       \hb@xt@\textwidth{%
5165         \hskip\columnwidth
5166         \hfil
5167         {\normalcolor\vrule \@width\columnseprule}%
5168         \hfil
5169         \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
5170         \hskip-\textwidth
5171         \hb@xt@\columnwidth{\box\@outputbox \hss}%
5172         \hskip\columnsep
5173         \hskip\columnwidth}}}%
5174   {}
5175 <@Footnote changes@>
5176 \IfBabelLayout{footnotes}%
5177   {\BabelFootnote\footnote\languagename{}}}%

```

```

5178 \BabelFootnote\localfootnote\language\language{}{}%
5179 \BabelFootnote\mainfootnote{}{}{}
5180 {}

```

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.

```

5181 \IfBabelLayout{counters*}%
5182 {\bbl@add\bbl@opt@layout{.counters.}%
5183 \AddToHook{shipout/before}{%
5184 \let\bbl@tempa\babelsublr
5185 \let\babelsublr\@firstofone
5186 \let\bbl@save@thepage\thepage
5187 \protected@edef\thepage{\thepage}%
5188 \let\babelsublr\bbl@tempa}%
5189 \AddToHook{shipout/after}{%
5190 \let\thepage\bbl@save@thepage}}{}
5191 \IfBabelLayout{counters}%
5192 {\let\bbl@latinarabic=\@arabic
5193 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5194 \let\bbl@asciroman=\@roman
5195 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5196 \let\bbl@asciiRoman=\@Roman
5197 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
5198 \fi % end if layout
5199 </xetex | texpet)

```

10.2 8-bit TeX

Which start just above, because some code is shared with xetex. Now, 8-bit specific stuff. If just one encoding has been declared, then assume no switching is necessary (1).

```

5200 (*texpet)
5201 \def\bbl@provide@extra#1{%
5202 % == auto-select encoding ==
5203 \ifx\bbl@encoding@select@off\empty\else
5204 \bbl@ifunset\bbl@encoding@#1{%
5205 {\def\@elt##1{##1,%
5206 \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5207 \count@\z@
5208 \bbl@foreach\bbl@tempe{%
5209 \def\bbl@tempd{##1}% Save last declared
5210 \advance\count@\@ne}%
5211 \ifnum\count@>\@ne % (1)
5212 \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5213 \ifx\bbl@tempa\relax \let\bbl@tempa\empty \fi
5214 \bbl@replace\bbl@tempa{ },}%
5215 \global\bbl@csarg\let{encoding@#1}\empty
5216 \bbl@xin@{\bbl@tempd,}{\bbl@tempa,}%
5217 \ifin@else % if main encoding included in ini, do nothing
5218 \let\bbl@tempb\relax
5219 \bbl@foreach\bbl@tempa{%
5220 \ifx\bbl@tempb\relax
5221 \bbl@xin@{##1,}{\bbl@tempa,}%
5222 \ifin@def\bbl@tempb{##1}\fi
5223 \fi}%
5224 \ifx\bbl@tempb\relax\else
5225 \bbl@exp{%
5226 \global\<bbl@add>\<bbl@preextras@#1>{\<bbl@encoding@#1>}%
5227 \gdef\<bbl@encoding@#1>{%
5228 \\babel@save\\f@encoding
5229 \\bbl@add\\originalTeX{\\selectfont}%
5230 \\fontencoding{\bbl@tempb}%
5231 \\selectfont}}%
5232 \fi

```

```

5233     \fi
5234     \fi}%
5235     {}%
5236     \fi}
5237 </texet>

```

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`).

```

5238 <!*luatex>
5239 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
5240 \bbl@trace{Read language.dat}
5241 \ifx\bbl@readstream\undefined
5242   \csname newread\endcsname\bbl@readstream
5243 \fi
5244 \begingroup
5245   \toks@{}
5246   \count@z@ % 0=start, 1=0th, 2=normal
5247   \def\bbl@process@line#1#2 #3 #4 {%
5248     \ifx=#1%
5249       \bbl@process@synonym{#2}%
5250     \else
5251       \bbl@process@language{#1#2}{#3}{#4}%
5252     \fi
5253     \ignorespaces}
5254   \def\bbl@manylang{%
5255     \ifnum\bbl@last>\@ne
5256       \bbl@info{Non-standard hyphenation setup}%
5257     \fi
5258     \let\bbl@manylang\relax}
5259   \def\bbl@process@language#1#2#3{%
5260     \ifcase\count@

```

```

5261     \ifundefined{zth@#1}{\count@tw@}{\count@one}%
5262 \or
5263     \count@tw@
5264 \fi
5265 \ifnum\count@=\tw@
5266     \expandafter\addlanguage\csname l@#1\endcsname
5267     \language\allocationnumber
5268     \chardef\bb@last\allocationnumber
5269     \bb@manylang
5270     \let\bb@elt\relax
5271     \xdef\bb@languages{%
5272         \bb@languages\bb@elt{#1}{\the\language}{#2}{#3}}%
5273 \fi
5274 \the\toks@
5275 \toks@{}}
5276 \def\bb@process@synonym@aux#1#2{%
5277     \global\expandafter\chardef\csname l@#1\endcsname#2\relax
5278     \let\bb@elt\relax
5279     \xdef\bb@languages{%
5280         \bb@languages\bb@elt{#1}{#2}{}}}%
5281 \def\bb@process@synonym#1{%
5282     \ifcase\count@
5283     \toks@\expandafter{\the\toks@\relax\bb@process@synonym{#1}}%
5284 \or
5285     \ifundefined{zth@#1}{\bb@process@synonym@aux{#1}{0}}}%
5286 \else
5287     \bb@process@synonym@aux{#1}{\the\bb@last}%
5288 \fi}
5289 \ifx\bb@languages\@undefined % Just a (sensible?) guess
5290     \chardef\l@english\z@
5291     \chardef\l@USenglish\z@
5292     \chardef\bb@last\z@
5293     \global\@namedef{bb@hyphendata@0}{{hyphen.tex}}
5294     \gdef\bb@languages{%
5295         \bb@elt{english}{0}{hyphen.tex}}%
5296         \bb@elt{USenglish}{0}{}}
5297 \else
5298     \global\let\bb@languages@format\bb@languages
5299     \def\bb@elt#1#2#3#4{% Remove all except language 0
5300         \ifnum#2>\z@\else
5301             \noexpand\bb@elt{#1}{#2}{#3}{#4}%
5302         \fi}%
5303     \xdef\bb@languages{\bb@languages}%
5304 \fi
5305 \def\bb@elt#1#2#3#4{\@namedef{zth@#1}} % Define flags
5306 \bb@languages
5307 \openin\bb@readstream=language.dat
5308 \ifeof\bb@readstream
5309     \bb@warning{I couldn't find language.dat. No additional\\%
5310         patterns loaded. Reported}%
5311 \else
5312     \loop
5313         \endlinechar\m@ne
5314         \read\bb@readstream to \bb@line
5315         \endlinechar\^^M
5316         \if T\ifeof\bb@readstream F\fi T\relax
5317         \ifx\bb@line\@empty\else
5318             \edef\bb@line{\bb@line\space\space\space}%
5319             \expandafter\bb@process@line\bb@line\relax
5320         \fi
5321     \repeat
5322 \fi
5323 \closein\bb@readstream

```



```

5324 \endgroup
5325 \bbl@trace{Macros for reading patterns files}
5326 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc#2}}
5327 \ifx\babelcatcodetablenum\undefined
5328 \ifx\newcatcodetable\undefined
5329 \def\babelcatcodetablenum{5211}
5330 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5331 \else
5332 \newcatcodetable\babelcatcodetablenum
5333 \newcatcodetable\bbl@pattcodes
5334 \fi
5335 \else
5336 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5337 \fi
5338 \def\bbl@luapatterns#1#2{%
5339 \bbl@get@enc#1:.\@@@
5340 \setbox\z@\hbox\bgroup
5341 \begingroup
5342 \savecatcodetable\babelcatcodetablenum\relax
5343 \initcatcodetable\bbl@pattcodes\relax
5344 \catcodetable\bbl@pattcodes\relax
5345 \catcode\#=6 \catcode\$_=3 \catcode\&=4 \catcode\^=7
5346 \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\~=13
5347 \catcode\@=11 \catcode\^I=10 \catcode\^J=12
5348 \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
5349 \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
5350 \catcode\`=12 \catcode\'=12 \catcode\"=12
5351 \input #1\relax
5352 \catcodetable\babelcatcodetablenum\relax
5353 \endgroup
5354 \def\bbl@tempa{#2}%
5355 \ifx\bbl@tempa\empty\else
5356 \input #2\relax
5357 \fi
5358 \egroup}%
5359 \def\bbl@patterns@lua#1{%
5360 \language=\expandafter\ifx\csname l@#1:f@encoding\endcsname\relax
5361 \csname l@#1\endcsname
5362 \edef\bbl@tempa{#1}%
5363 \else
5364 \csname l@#1:f@encoding\endcsname
5365 \edef\bbl@tempa{#1:f@encoding}%
5366 \fi\relax
5367 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5368 \@ifundefined{bbl@hyphendata@the\language}%
5369 {\def\bbl@elt##1##2##3##4{%
5370 \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
5371 \def\bbl@tempb{##3}%
5372 \ifx\bbl@tempb\empty\else % if not a synonymous
5373 \def\bbl@tempc{##3}{##4}}%
5374 \fi
5375 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5376 \fi}%
5377 \bbl@languages
5378 \@ifundefined{bbl@hyphendata@the\language}%
5379 {\bbl@info{No hyphenation patterns were set for\\%
5380 language '\bbl@tempa'. Reported}}%
5381 {\expandafter\expandafter\expandafter\bbl@luapatterns
5382 \csname bbl@hyphendata@the\language\endcsname}}}}
5383 \endinput\fi

```

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

```

5384 \ifx\DisableBabelHook\undefined

```

```

5385 \AddBabelHook{luatex}{everylanguage}{%
5386   \def\process@language##1##2##3{%
5387     \def\process@line####1####2 ####3 ####4 {}}
5388 \AddBabelHook{luatex}{loadpatterns}{%
5389   \input #1\relax
5390   \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5391     {#{1}{}}
5392 \AddBabelHook{luatex}{loadexceptions}{%
5393   \input #1\relax
5394   \def\bbl@tempb##1##2{#{#1}{#1}}%
5395   \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5396     {\expandafter\expandafter\expandafter\bbl@tempb
5397       \csname bbl@hyphendata@the\language\endcsname}}
5398 \endinput\fi

```

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

```

5399 \begingroup % TODO - to a lua file
5400 \catcode`\%=12
5401 \catcode`\'=12
5402 \catcode`\ "=12
5403 \catcode`\:=12
5404 \directlua{
5405   Babel = Babel or {}
5406   function Babel.lua_error(e, a)
5407     tex.print([[ \noexpand\csname bbl@error\endcsname{]] ..
5408       e .. '}' .. (a or '') .. '}{ }')
5409   end
5410   function Babel.bytes(line)
5411     return line:gsub(".",
5412       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5413   end
5414   function Babel.begin_process_input()
5415     if luatexbase and luatexbase.add_to_callback then
5416       luatexbase.add_to_callback('process_input_buffer',
5417         Babel.bytes, 'Babel.bytes')
5418     else
5419       Babel.callback = callback.find('process_input_buffer')
5420       callback.register('process_input_buffer', Babel.bytes)
5421     end
5422   end
5423   function Babel.end_process_input ()
5424     if luatexbase and luatexbase.remove_from_callback then
5425       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5426     else
5427       callback.register('process_input_buffer', Babel.callback)
5428     end
5429   end
5430   function Babel.addpatterns(pp, lg)
5431     local lg = lang.new(lg)
5432     local pats = lang.patterns(lg) or ''
5433     lang.clear_patterns(lg)
5434     for p in pp:gmatch('[^%s]+') do
5435       ss = ''
5436       for i in string.utfcharacters(p:gsub('%d', '')) do
5437         ss = ss .. '%d?' .. i
5438       end
5439       ss = ss:gsub('^%d%?%', '%%.') .. '%d?'
5440       ss = ss:gsub('%.%d%?$', '%%.')
5441       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5442       if n == 0 then
5443         tex.sprint(
5444           [[ \string\csname\space bbl@info\endcsname{New pattern: }]]

```

```

5445     .. p .. [{}])
5446     pats = pats .. ' ' .. p
5447     else
5448         tex.sprint(
5449             [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5450             .. p .. [{}])
5451     end
5452 end
5453 lang.patterns(lg, pats)
5454 end
5455 Babel.characters = Babel.characters or {}
5456 Babel.ranges = Babel.ranges or {}
5457 function Babel.hlist_has_bidi(head)
5458     local has_bidi = false
5459     local ranges = Babel.ranges
5460     for item in node.traverse(head) do
5461         if item.id == node.id'glyph' then
5462             local itemchar = item.char
5463             local chardata = Babel.characters[itemchar]
5464             local dir = chardata and chardata.d or nil
5465             if not dir then
5466                 for nn, et in ipairs(ranges) do
5467                     if itemchar < et[1] then
5468                         break
5469                     elseif itemchar <= et[2] then
5470                         dir = et[3]
5471                         break
5472                     end
5473                 end
5474             end
5475             if dir and (dir == 'al' or dir == 'r') then
5476                 has_bidi = true
5477             end
5478         end
5479     end
5480     return has_bidi
5481 end
5482 function Babel.set_chranges_b (script, chrng)
5483     if chrng == '' then return end
5484     texio.write('Replacing ' .. script .. ' script ranges')
5485     Babel.script_blocks[script] = {}
5486     for s, e in string.gmatch(chrng..' ', '(.)%.%.(-)%s') do
5487         table.insert(
5488             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5489     end
5490 end
5491 function Babel.discard_sublrl(str)
5492     if str:find( [[\string\indexentry]] ) and
5493         str:find( [[\string\babelsublrl]] ) then
5494         str = str:gsub( [[\string\babelsublrl%*{%b}]]),
5495             function(m) return m:sub(2,-2) end )
5496     end
5497     return str
5498 end
5499 }
5500 \endgroup
5501 \ifx\newattribute\@undefined\else % Test for plain
5502     \newattribute\bbl@attr@locale
5503     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5504     \AddBabelHook{luatex}{beforeextras}{%
5505         \setattribute\bbl@attr@locale\localeid}
5506 \fi
5507 \def\BabelStringsDefault{unicode}

```

```

5508 \let\luabbl@stop\relax
5509 \AddBabelHook{luatex}{encodedcommands}{%
5510   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5511   \ifx\bbl@tempa\bbl@tempb\else
5512     \directlua{Babel.begin_process_input()}%
5513     \def\luabbl@stop{%
5514       \directlua{Babel.end_process_input()}%
5515     \fi}%
5516 \AddBabelHook{luatex}{stopcommands}{%
5517   \luabbl@stop
5518   \let\luabbl@stop\relax}
5519 \AddBabelHook{luatex}{patterns}{%
5520   \@ifundefined{bbl@hyphendata@the\language}%
5521     {\def\bbl@elt##1##2##3##4{%
5522       \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5523       \def\bbl@tempb{##3}%
5524       \ifx\bbl@tempb\@empty\else % if not a synonymous
5525         \def\bbl@tempc{{##3}{##4}}%
5526       \fi
5527       \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5528     \fi}%
5529   \bbl@languages
5530   \@ifundefined{bbl@hyphendata@the\language}%
5531     {\bbl@info{No hyphenation patterns were set for\%
5532       language '#2'. Reported}}%
5533     {\expandafter\expandafter\expandafter\bbl@luapatterns
5534       \csname bbl@hyphendata@the\language\endcsname}}}%
5535   \@ifundefined{bbl@patterns@}{%
5536     \begingroup
5537     \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5538     \ifin@else
5539       \ifx\bbl@patterns@\@empty\else
5540         \directlua{ Babel.addpatterns(
5541           [[\bbl@patterns@]], \number\language) }%
5542       \fi
5543       \@ifundefined{bbl@patterns@#1}%
5544         \@empty
5545         {\directlua{ Babel.addpatterns(
5546           [[\space\csname bbl@patterns@#1\endcsname]],
5547           \number\language) }}%
5548       \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5549     \fi
5550   \endgroup}%
5551   \bbl@exp{%
5552     \bbl@ifunset{bbl@prehc@languagename}{%
5553       {\bbl@ifblank{\bbl@cs{prehc@languagename}}}%
5554       {\prehyphenchar=\bbl@cl{prehc}\relax}}}}

```

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

```

5555 \@onlypreamble\babelpatterns
5556 \AtEndOfPackage{%
5557   \newcommand\babelpatterns[2][\@empty]{%
5558     \ifx\bbl@patterns@\relax
5559       \let\bbl@patterns@\@empty
5560     \fi
5561     \ifx\bbl@pttnlist\@empty\else
5562       \bbl@warning{%
5563         You must not intermingle \string\selectlanguage\space and\%
5564         \string\babelpatterns\space or some patterns will not\%
5565         be taken into account. Reported}%
5566     \fi

```

```

5567 \ifx\@empty#1%
5568 \protected@edef\bbbl@patterns@{\bbbl@patterns@space#2}%
5569 \else
5570 \edef\bbbl@tempb{\zap@space#1 \@empty}%
5571 \bbbl@for\bbbl@tempa\bbbl@tempb{%
5572 \bbbl@fixname\bbbl@tempa
5573 \bbbl@iflanguage\bbbl@tempa{%
5574 \bbbl@csarg\protected@edef{patterns@\bbbl@tempa}{%
5575 \@ifundefined{bbbl@patterns@\bbbl@tempa}%
5576 \@empty
5577 {\csname bbl@patterns@\bbbl@tempa\endcsname\space}%
5578 #2}}}%
5579 \fi}}

```

10.4 Southeast Asian scripts

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

```

5580 % TODO - to a lua file -- or a logical place
5581 \directlua{
5582   Babel = Babel or {}
5583   Babel.linebreaking = Babel.linebreaking or {}
5584   Babel.linebreaking.before = {}
5585   Babel.linebreaking.after = {}
5586   Babel.locale = {} % Free to use, indexed by \localeid
5587   function Babel.linebreaking.add_before(func, pos)
5588     tex.print([[ \noexpand\csname bbl@lua-hyphenate\endcsname]])
5589     if pos == nil then
5590       table.insert(Babel.linebreaking.before, func)
5591     else
5592       table.insert(Babel.linebreaking.before, pos, func)
5593     end
5594   end
5595   function Babel.linebreaking.add_after(func)
5596     tex.print([[ \noexpand\csname bbl@lua-hyphenate\endcsname]])
5597     table.insert(Babel.linebreaking.after, func)
5598   end
5599 }
5600 \def\bbbl@intraspace#1 #2 #3\@@{%
5601 \directlua{
5602   Babel = Babel or {}
5603   Babel.intraspaces = Babel.intraspaces or {}
5604   Babel.intraspaces['\csname bbl@sbc-p@\language-name\endcsname'] = %
5605     {b = #1, p = #2, m = #3}
5606   Babel.locale_props[\the\localeid].intraspace = %
5607     {b = #1, p = #2, m = #3}
5608 }}
5609 \def\bbbl@intrapenalty#1\@@{%
5610 \directlua{
5611   Babel = Babel or {}
5612   Babel.intrapenalties = Babel.intrapenalties or {}
5613   Babel.intrapenalties['\csname bbl@sbc-p@\language-name\endcsname'] = #1
5614   Babel.locale_props[\the\localeid].intrapenalty = #1
5615 }}
5616 \beginingroup
5617 \catcode`\%=12
5618 \catcode`\&=14
5619 \catcode`\'=12
5620 \catcode`\~ =12
5621 \gdef\bbbl@seaintraspace{&
5622 \let\bbbl@seaintraspace\relax

```

```

5623 \directlua{
5624   Babel = Babel or {}
5625   Babel.sea_enabled = true
5626   Babel.sea_ranges = Babel.sea_ranges or {}
5627   function Babel.set_chrngs (script, chrng)
5628     local c = 0
5629     for s, e in string.gmatch(chrng..' ', '(-)%.%(-)%s') do
5630       Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5631       c = c + 1
5632     end
5633   end
5634   function Babel.sea_disc_to_space (head)
5635     local sea_ranges = Babel.sea_ranges
5636     local last_char = nil
5637     local quad = 655360      &% 10 pt = 655360 = 10 * 65536
5638     for item in node.traverse(head) do
5639       local i = item.id
5640       if i == node.id'glyph' then
5641         last_char = item
5642       elseif i == 7 and item.subtype == 3 and last_char
5643         and last_char.char > 0x0C99 then
5644         quad = font.getfont(last_char.font).size
5645         for lg, rg in pairs(sea_ranges) do
5646           if last_char.char > rg[1] and last_char.char < rg[2] then
5647             lg = lg:sub(1, 4) &% Remove trailing number of, eg, Cyril1
5648             local intraspace = Babel.intraspaces[lg]
5649             local intrapenalty = Babel.intrapanalties[lg]
5650             local n
5651             if intrapenalty ~= 0 then
5652               n = node.new(14, 0) &% penalty
5653               n.penalty = intrapenalty
5654               node.insert_before(head, item, n)
5655             end
5656             n = node.new(12, 13) &% (glue, spaceskip)
5657             node.setglue(n, intraspace.b * quad,
5658               intraspace.p * quad,
5659               intraspace.m * quad)
5660             node.insert_before(head, item, n)
5661             node.remove(head, item)
5662           end
5663         end
5664       end
5665     end
5666   end
5667 }&
5668 \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.

```

5669 \catcode`\%=14
5670 \gdef\bbl@cjk intraspace{%
5671   \let\bbl@cjk intraspace\relax
5672   \directlua{
5673     Babel = Babel or {}
5674     require('babel-data-cjk.lua')
5675     Babel.cjk_enabled = true
5676     function Babel.cjk_linebreak(head)

```

```

5677     local GLYPH = node.id'glyph'
5678     local last_char = nil
5679     local quad = 655360      % 10 pt = 655360 = 10 * 65536
5680     local last_class = nil
5681     local last_lang = nil
5682
5683     for item in node.traverse(head) do
5684         if item.id == GLYPH then
5685
5686             local lang = item.lang
5687
5688             local LOCALE = node.get_attribute(item,
5689                 Babel.attr_locale)
5690             local props = Babel.locale_props[LOCALE]
5691
5692             local class = Babel.cjk_class[item.char].c
5693
5694             if props.cjk_quotes and props.cjk_quotes[item.char] then
5695                 class = props.cjk_quotes[item.char]
5696             end
5697
5698             if class == 'cp' then class = 'cl' % ]] as CL
5699             elseif class == 'id' then class = 'I'
5700             elseif class == 'cj' then class = 'I' % loose
5701             end
5702
5703             local br = 0
5704             if class and last_class and Babel.cjk_breaks[last_class][class] then
5705                 br = Babel.cjk_breaks[last_class][class]
5706             end
5707
5708             if br == 1 and props.linebreak == 'c' and
5709                 lang ~= \the\l@nohyphenation\space and
5710                 last_lang ~= \the\l@nohyphenation then
5711                 local intrapenalty = props.intrapenalty
5712                 if intrapenalty ~= 0 then
5713                     local n = node.new(14, 0)      % penalty
5714                     n.penalty = intrapenalty
5715                     node.insert_before(head, item, n)
5716                 end
5717                 local intraspace = props.intraspace
5718                 local n = node.new(12, 13)      % (glue, spaceskip)
5719                 node.setglue(n, intraspace.b * quad,
5720                     intraspace.p * quad,
5721                     intraspace.m * quad)
5722                 node.insert_before(head, item, n)
5723             end
5724
5725             if font.getfont(item.font) then
5726                 quad = font.getfont(item.font).size
5727             end
5728             last_class = class
5729             last_lang = lang
5730             else % if penalty, glue or anything else
5731                 last_class = nil
5732             end
5733         end
5734         lang.hyphenate(head)
5735     end
5736 }%
5737 \bbl@luahyphenate}
5738 \gdef\bbl@luahyphenate{%
5739 \let\bbl@luahyphenate\relax

```

```

5740 \directlua{
5741   luatexbase.add_to_callback('hyphenate',
5742   function (head, tail)
5743     if Babel.linebreaking.before then
5744       for k, func in ipairs(Babel.linebreaking.before) do
5745         func(head)
5746       end
5747     end
5748     lang.hyphenate(head)
5749     if Babel.cjk_enabled then
5750       Babel.cjk_linebreak(head)
5751     end
5752     if Babel.linebreaking.after then
5753       for k, func in ipairs(Babel.linebreaking.after) do
5754         func(head)
5755       end
5756     end
5757     if Babel.sea_enabled then
5758       Babel.sea_disc_to_space(head)
5759     end
5760   end,
5761   'Babel.hyphenate')
5762 }
5763 }
5764 \endgroup
5765 \def\bbl@provide@intraspace{%
5766   \bbl@ifunset{\bbl@intsp@languagename}{}%
5767   {\expandafter\ifx\csname bbl@intsp@languagename\endcsname\@empty\else
5768     \bbl@xin{/c}{/\bbl@cl{lbrk}}%
5769     \ifin@           % cjk
5770     \bbl@cjk@intraspace
5771     \directlua{
5772       Babel = Babel or {}
5773       Babel.locale_props = Babel.locale_props or {}
5774       Babel.locale_props[\the\localeid].linebreak = 'c'
5775     }%
5776     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@}%
5777     \ifx\bbl@KVP@intrapenalty\@nnil
5778       \bbl@intrapenalty0\@@
5779     \fi
5780   \else           % sea
5781     \bbl@sea@intraspace
5782     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@}%
5783     \directlua{
5784       Babel = Babel or {}
5785       Babel.sea_ranges = Babel.sea_ranges or {}
5786       Babel.set_chranges('\bbl@cl{sbc}',
5787         '\bbl@cl{chrng}')
5788     }%
5789     \ifx\bbl@KVP@intrapenalty\@nnil
5790       \bbl@intrapenalty0\@@
5791     \fi
5792   \fi
5793 \fi
5794 \ifx\bbl@KVP@intrapenalty\@nnil\else
5795   \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5796 \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-`

```
5797 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
```



```

5798 \def\bblar@chars{%
5799   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5800   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5801   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5802 \def\bblar@elongated{%
5803   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5804   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5805   0649,064A}
5806 \begingroup
5807   \catcode`_ =11 \catcode`:=11
5808   \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5809 \endgroup
5810 \gdef\bbl@arabicjust{% TODO. Allow for several locales.
5811   \let\bbl@arabicjust\relax
5812   \newattribute\bblar@kashida
5813   \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5814   \bblar@kashida=\z@
5815   \bbl@patchfont{\bbl@parsejalt}}%
5816   \directlua{
5817     Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5818     Babel.arabic.elong_map[\the\localeid] = {}
5819     luatexbase.add_to_callback('post_linebreak_filter',
5820       Babel.arabic.justify, 'Babel.arabic.justify')
5821     luatexbase.add_to_callback('hpack_filter',
5822       Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5823   }}%

```

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

```

5824 \def\bblar@fetchjalt#1#2#3#4{%
5825   \bbl@exp{\bbl@foreach{#1}}{%
5826     \bbl@ifunset{bblar@JE@##1}%
5827     {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"##1#2}}%
5828     {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"@nameuse{bblar@JE@##1#2}}}%
5829   \directlua{%
5830     local last = nil
5831     for item in node.traverse(tex.box[0].head) do
5832       if item.id == node.id'glyph' and item.char > 0x600 and
5833         not (item.char == 0x200D) then
5834         last = item
5835       end
5836     end
5837     Babel.arabic.#3['##1#4'] = last.char
5838   }}

```

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?

```

5839 \gdef\bbl@parsejalt{%
5840   \ifx\addfontfeature\undefined\else
5841     \bbl@xin@{/e}{\bbl@cl{\lnbrk}}%
5842     \ifin@
5843     \directlua{%
5844       if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5845         Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5846         tex.print([[string\cname\space bbl@parsejalti\endcname]])
5847       end
5848     }%
5849     \fi
5850   \fi}
5851 \gdef\bbl@parsejalti{%
5852   \begingroup
5853     \let\bbl@parsejalt\relax % To avoid infinite loop
5854     \edef\bbl@tempb{\fontid\font}%
5855     \bblar@nofswarn
5856     \bblar@fetchjalt\bblar@elongated}{\from}}%

```

```

5857 \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
5858 \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
5859 \addfontfeature{RawFeature+=jalt}%
5860 % \namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5861 \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5862 \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
5863 \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
5864 \directlua{%
5865     for k, v in pairs(Babel.arabic.from) do
5866         if Babel.arabic.dest[k] and
5867             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5868             Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5869                 [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5870         end
5871     end
5872 }%
5873 \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```

5874 \begingroup
5875 \catcode`#=11
5876 \catcode`~ =11
5877 \directlua{
5878
5879 Babel.arabic = Babel.arabic or {}
5880 Babel.arabic.from = {}
5881 Babel.arabic.dest = {}
5882 Babel.arabic.justify_factor = 0.95
5883 Babel.arabic.justify_enabled = true
5884 Babel.arabic.kashida_limit = -1
5885
5886 function Babel.arabic.justify(head)
5887     if not Babel.arabic.justify_enabled then return head end
5888     for line in node.traverse_id(node.id'hlist', head) do
5889         Babel.arabic.justify_hlist(head, line)
5890     end
5891     return head
5892 end
5893
5894 function Babel.arabic.justify_hbox(head, gc, size, pack)
5895     local has_inf = false
5896     if Babel.arabic.justify_enabled and pack == 'exactly' then
5897         for n in node.traverse_id(12, head) do
5898             if n.stretch_order > 0 then has_inf = true end
5899         end
5900         if not has_inf then
5901             Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5902         end
5903     end
5904     return head
5905 end
5906
5907 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5908     local d, new
5909     local k_list, k_item, pos_inline
5910     local width, width_new, full, k_curr, wt_pos, goal, shift
5911     local subst_done = false
5912     local elong_map = Babel.arabic.elong_map
5913     local cnt
5914     local last_line
5915     local GLYPH = node.id'glyph'
5916     local KASHIDA = Babel.attr_kashida
5917     local LOCALE = Babel.attr_locale

```

```

5918
5919 if line == nil then
5920   line = {}
5921   line.glue_sign = 1
5922   line.glue_order = 0
5923   line.head = head
5924   line.shift = 0
5925   line.width = size
5926 end
5927
5928 % Exclude last line. todo. But-- it discards one-word lines, too!
5929 % ? Look for glue = 12:15
5930 if (line.glue_sign == 1 and line.glue_order == 0) then
5931   elongs = {}      % Stores elongated candidates of each line
5932   k_list = {}      % And all letters with kashida
5933   pos_inline = 0  % Not yet used
5934
5935   for n in node.traverse_id(GLYPH, line.head) do
5936     pos_inline = pos_inline + 1 % To find where it is. Not used.
5937
5938     % Elongated glyphs
5939     if elong_map then
5940       local locale = node.get_attribute(n, LOCALE)
5941       if elong_map[locale] and elong_map[locale][n.font] and
5942         elong_map[locale][n.font][n.char] then
5943         table.insert(elongs, {node = n, locale = locale} )
5944         node.set_attribute(n.prev, KASHIDA, 0)
5945       end
5946     end
5947
5948     % Tatwil
5949     if Babel.kashida_wts then
5950       local k_wt = node.get_attribute(n, KASHIDA)
5951       if k_wt > 0 then % todo. parameter for multi inserts
5952         table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5953       end
5954     end
5955
5956   end % of node.traverse_id
5957
5958   if #elongs == 0 and #k_list == 0 then goto next_line end
5959   full = line.width
5960   shift = line.shift
5961   goal = full * Babel.arabic.justify_factor % A bit crude
5962   width = node.dimensions(line.head) % The 'natural' width
5963
5964   % == Elongated ==
5965   % Original idea taken from 'chickenize'
5966   while (#elongs > 0 and width < goal) do
5967     subst_done = true
5968     local x = #elongs
5969     local curr = elongs[x].node
5970     local oldchar = curr.char
5971     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5972     width = node.dimensions(line.head) % Check if the line is too wide
5973     % Substitute back if the line would be too wide and break:
5974     if width > goal then
5975       curr.char = oldchar
5976     end
5977   end
5978   % If continue, pop the just substituted node from the list:
5979   table.remove(elongs, x)
5980 end

```

```

5981
5982 % == Tatwil ==
5983 if #k_list == 0 then goto next_line end
5984
5985 width = node.dimensions(line.head) % The 'natural' width
5986 k_curr = #k_list % Traverse backwards, from the end
5987 wt_pos = 1
5988
5989 while width < goal do
5990     subst_done = true
5991     k_item = k_list[k_curr].node
5992     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5993         d = node.copy(k_item)
5994         d.char = 0x0640
5995         d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
5996         d.xoffset = 0
5997         line.head, new = node.insert_after(line.head, k_item, d)
5998         width_new = node.dimensions(line.head)
5999         if width > goal or width == width_new then
6000             node.remove(line.head, new) % Better compute before
6001             break
6002         end
6003         if Babel.fix_diacr then
6004             Babel.fix_diacr(k_item.next)
6005         end
6006         width = width_new
6007     end
6008     if k_curr == 1 then
6009         k_curr = #k_list
6010         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
6011     else
6012         k_curr = k_curr - 1
6013     end
6014 end
6015
6016 % Limit the number of tatweel by removing them. Not very efficient,
6017 % but it does the job in a quite predictable way.
6018 if Babel.arabic.kashida_limit > -1 then
6019     cnt = 0
6020     for n in node.traverse_id(GLYPH, line.head) do
6021         if n.char == 0x0640 then
6022             cnt = cnt + 1
6023             if cnt > Babel.arabic.kashida_limit then
6024                 node.remove(line.head, n)
6025             end
6026         else
6027             cnt = 0
6028         end
6029     end
6030 end
6031
6032 ::next_line::
6033
6034 % Must take into account marks and ins, see luatex manual.
6035 % Have to be executed only if there are changes. Investigate
6036 % what's going on exactly.
6037 if subst_done and not gc then
6038     d = node.hpack(line.head, full, 'exactly')
6039     d.shift = shift
6040     node.insert_before(head, line, d)
6041     node.remove(head, line)
6042 end
6043 end % if process line

```

```

6044 end
6045 }
6046 \endgroup
6047 \fi\fi % ends Arabic just block: \ifnum\bbl@bidimode>100...

```

10.7 Common stuff

```
6048 <@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.

```

6049% TODO - to a lua file
6050 \directlua{
6051 Babel.script_blocks = {
6052   ['dflt'] = {},
6053   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
6054             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE0, 0x1EEF}},
6055   ['Armn'] = {{0x0530, 0x058F}},
6056   ['Beng'] = {{0x0980, 0x09FF}},
6057   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0ABBF}},
6058   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
6059   ['Cyrll'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
6060             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
6061   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
6062   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
6063             {0xAB00, 0xAB2F}},
6064   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
6065   % Don't follow strictly Unicode, which places some Coptic letters in
6066   % the 'Greek and Coptic' block
6067   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
6068   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
6069             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
6070             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
6071             {0x2000, 0x2A6DF}, {0x2A700, 0x2B73F},
6072             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
6073             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
6074   ['Hebr'] = {{0x0590, 0x05FF}},
6075   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
6076             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
6077   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
6078   ['Knda'] = {{0x0C80, 0x0CFF}},
6079   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
6080             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
6081             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
6082   ['Laoo'] = {{0x0E80, 0x0EFF}},
6083   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
6084             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
6085             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
6086   ['Mahj'] = {{0x1150, 0x117F}},
6087   ['Mlym'] = {{0x0D00, 0x0D7F}},
6088   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
6089   ['Orya'] = {{0x0B00, 0x0B7F}},
6090   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
6091   ['Sycr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
6092   ['Taml'] = {{0x0B80, 0x0BFF}},

```

```

6093 ['Telu'] = {{0x0C00, 0x0C7F}},
6094 ['Tfng'] = {{0x2D30, 0x2D7F}},
6095 ['Thai'] = {{0x0E00, 0x0E7F}},
6096 ['Tibt'] = {{0x0F00, 0x0FFF}},
6097 ['Vaii'] = {{0xA500, 0xA63F}},
6098 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
6099 }
6100
6101 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
6102 Babel.script_blocks.Hant = Babel.script_blocks.Hans
6103 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
6104
6105 function Babel.locale_map(head)
6106   if not Babel.locale_mapped then return head end
6107
6108   local LOCALE = Babel.attr_locale
6109   local GLYPH = node.id('glyph')
6110   local inmath = false
6111   local toloc_save
6112   for item in node.traverse(head) do
6113     local toloc
6114     if not inmath and item.id == GLYPH then
6115       % Optimization: build a table with the chars found
6116       if Babel.chr_to_loc[item.char] then
6117         toloc = Babel.chr_to_loc[item.char]
6118       else
6119         for lc, maps in pairs(Babel.loc_to_scr) do
6120           for _, rg in pairs(maps) do
6121             if item.char >= rg[1] and item.char <= rg[2] then
6122               Babel.chr_to_loc[item.char] = lc
6123               toloc = lc
6124               break
6125             end
6126           end
6127         end
6128         % Treat composite chars in a different fashion, because they
6129         % 'inherit' the previous locale.
6130         if (item.char >= 0x0300 and item.char <= 0x036F) or
6131            (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
6132            (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
6133           Babel.chr_to_loc[item.char] = -2000
6134           toloc = -2000
6135         end
6136         if not toloc then
6137           Babel.chr_to_loc[item.char] = -1000
6138         end
6139       end
6140       if toloc == -2000 then
6141         toloc = toloc_save
6142       elseif toloc == -1000 then
6143         toloc = nil
6144       end
6145       if toloc and Babel.locale_props[toloc] and
6146          Babel.locale_props[toloc].letters and
6147          tex.getcatcode(item.char) \string~= ll then
6148         toloc = nil
6149       end
6150       if toloc and Babel.locale_props[toloc].script
6151          and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6152          and Babel.locale_props[toloc].script ==
6153          Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6154         toloc = nil
6155       end

```

```

6156     if toloc then
6157         if Babel.locale_props[toloc].lg then
6158             item.lang = Babel.locale_props[toloc].lg
6159             node.set_attribute(item, LOCALE, toloc)
6160         end
6161         if Babel.locale_props[toloc]['/'..item.font] then
6162             item.font = Babel.locale_props[toloc]['/'..item.font]
6163         end
6164     end
6165     toloc_save = toloc
6166     elseif not inmath and item.id == 7 then % Apply recursively
6167         item.replace = item.replace and Babel.locale_map(item.replace)
6168         item.pre      = item.pre and Babel.locale_map(item.pre)
6169         item.post     = item.post and Babel.locale_map(item.post)
6170     elseif item.id == node.id'math' then
6171         inmath = (item.subtype == 0)
6172     end
6173 end
6174 return head
6175 end
6176 }

```

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

```

6177 \newcommand\babelcharproperty[1]{%
6178   \count@=#1\relax
6179   \ifvmode
6180     \expandafter\bbl@chprop
6181   \else
6182     \bbl@error{charproperty-only-vertical}{}{}{}%
6183   \fi}
6184 \newcommand\bbl@chprop[3][\the\count@]{%
6185   \@tempcnta=#1\relax
6186   \bbl@ifunset{bbl@chprop@#2}% {unknown-char-property}
6187   {\bbl@error{unknown-char-property}{}{#2}{}%
6188   }%
6189   \loop
6190     \bbl@cs{chprop@#2}{#3}%
6191   \ifnum\count@<\@tempcnta
6192     \advance\count@\@ne
6193   \repeat}
6194 \def\bbl@chprop@direction#1{%
6195   \directlua{
6196     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6197     Babel.characters[\the\count@]['d'] = '#1'
6198   }}
6199 \let\bbl@chprop@bc\bbl@chprop@direction
6200 \def\bbl@chprop@mirror#1{%
6201   \directlua{
6202     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6203     Babel.characters[\the\count@]['m'] = '\number#1'
6204   }}
6205 \let\bbl@chprop@bmg\bbl@chprop@mirror
6206 \def\bbl@chprop@linebreak#1{%
6207   \directlua{
6208     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6209     Babel.cjk_characters[\the\count@]['c'] = '#1'
6210   }}
6211 \let\bbl@chprop@lb\bbl@chprop@linebreak
6212 \def\bbl@chprop@locale#1{%
6213   \directlua{
6214     Babel.chr_to_loc = Babel.chr_to_loc or {}
6215     Babel.chr_to_loc[\the\count@] =

```

```
6216 \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
6217 }
```

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.

```
6218 \directlua{
6219   Babel.nohyphenation = \the\l@nohyphenation
6220 }
```

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).

```
6221 \begingroup
6222 \catcode`\~ = 12
6223 \catcode`\% = 12
6224 \catcode`\& = 14
6225 \catcode`\| = 12
6226 \gdef\babelprehyphenation{&%
6227   \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}}{]}
6228 \gdef\babelposthyphenation{&%
6229   \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}}{]}
6230 \gdef\bbl@settransform#1[#2]#3#4#5{&%
6231   \ifcase#1
6232     \bbl@activateprehyphen
6233   \or
6234     \bbl@activateposthyphen
6235   \fi
6236 \begingroup
6237   \def\babeltempa{\bbl@add@list\babeltempb}&%
6238   \let\babeltempb\@empty
6239   \def\bbl@tempa{#5}&%
6240   \bbl@replace\bbl@tempa{,}{ , }&% TODO. Ugly trick to preserve {}
6241   \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
6242     \bbl@ifsamestring{##1}{remove}&%
6243     {\bbl@add@list\babeltempb{nil}}&%
6244     {\directlua{
6245       local rep = [= [#1] =]
6246       rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6247       rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
6248       rep = rep:gsub('^%s*(after)%s*', 'after = true, ')
6249       rep = rep:gsub('(string)%s*=%s*([%s,]*)', Babel.capture_func)
6250       rep = rep:gsub('node%s*=%s*(%a)%s*(%a*)', Babel.capture_node)
6251       rep = rep:gsub(&%
6252         '(norule)%s*=%s*([%-d%.]+)%s+([%-d%.]+)%s+([%-d%.]+)',
6253         'norule = { ' .. '%2, %3, %4' .. ' }')
6254       if #1 == 0 or #1 == 2 then
6255         rep = rep:gsub(&%
6256           '(space)%s*=%s*([%-d%.]+)%s+([%-d%.]+)%s+([%-d%.]+)',
6257           'space = { ' .. '%2, %3, %4' .. ' }')
6258         rep = rep:gsub(&%
6259           '(spacefactor)%s*=%s*([%-d%.]+)%s+([%-d%.]+)%s+([%-d%.]+)',
6260           'spacefactor = { ' .. '%2, %3, %4' .. ' }')
6261         rep = rep:gsub('(kashida)%s*=%s*([%s,]*)', Babel.capture_kashida)
6262       else
6263         rep = rep:gsub(' (no)%s*=%s*([%s,]*)', Babel.capture_func)
6264         rep = rep:gsub(' (pre)%s*=%s*([%s,]*)', Babel.capture_func)
6265         rep = rep:gsub(' (post)%s*=%s*([%s,]*)', Babel.capture_func)
6266       end

```



```

6267         tex.print([[string\babeltempa{[]} .. rep .. [{}]])
6268     }]}&%
6269 \bbl@foreach\babeltempb{&%
6270     \bbl@forkv{##1}}{&%
6271         \in{,###1,}{,nil,step,data,remove,insert,string,no,pre,no,&%
6272         post,penalty,kashida,space,spacefactor,kern,node,after,norule,}&%
6273         \ifin@else
6274         \bbl@error{bad-transform-option}{###1}{}}{&%
6275         \fi}}&%
6276 \let\bbl@kv@attribute\relax
6277 \let\bbl@kv@label\relax
6278 \let\bbl@kv@fonts\@empty
6279 \bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}&%
6280 \ifx\bbl@kv@fonts\@empty\else\bbl@settransfont\fi
6281 \ifx\bbl@kv@attribute\relax
6282     \ifx\bbl@kv@label\relax\else
6283         \bbl@exp{\bbl@trim@def\bbl@kv@fonts{\bbl@kv@fonts}}&%
6284         \bbl@replace\bbl@kv@fonts{ }{,}&%
6285         \edef\bbl@kv@attribute{\bbl@ATR@\bbl@kv@label @#3@\bbl@kv@fonts}&%
6286         \count@ \z@
6287         \def\bbl@elt##1##2##3{&%
6288             \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
6289             {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
6290             {\count@\@ne}&%
6291             {\bbl@error{font-conflict-transforms}{}}{}}}&%
6292             {}&%
6293             \bbl@transfont@list
6294             \ifnum\count@=\z@
6295                 \bbl@exp{\global\bbl@add\bbl@transfont@list
6296                     {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
6297             \fi
6298             \bbl@ifunset{\bbl@kv@attribute}&%
6299             {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6300             {}&%
6301             \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
6302             \fi
6303         \else
6304             \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6305             \fi
6306         \directlua{
6307             local lbrk = Babel.linebreaking.replacements[#1]
6308             local u = unicode.utf8
6309             local id, attr, label
6310             if #1 == 0 then
6311                 id = \the\csname bbl@id@#3\endcsname\space
6312             else
6313                 id = \the\csname l@#3\endcsname\space
6314             end
6315             \ifx\bbl@kv@attribute\relax
6316                 attr = -1
6317             \else
6318                 attr = luatexbase.registernumber'\bbl@kv@attribute'
6319             \fi
6320             \ifx\bbl@kv@label\relax\else &% Same refs:
6321                 label = [==[\bbl@kv@label]==]
6322             \fi
6323             &% Convert pattern:
6324             local patt = string.gsub([==[#4]==], '%s', '')
6325             if #1 == 0 then
6326                 patt = string.gsub(patt, '|', ' ')
6327             end
6328             if not u.find(patt, '()', nil, true) then
6329                 patt = '()' .. patt .. '()'

```

```

6330     end
6331     if #1 == 1 then
6332         patt = string.gsub(patt, '%(%)%^\', '^()')
6333         patt = string.gsub(patt, '%$(%)', '()$')
6334     end
6335     patt = u.gsub(patt, '{(.)}',
6336         function (n)
6337             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6338         end)
6339     patt = u.gsub(patt, '{(%x%x%x%x+)}',
6340         function (n)
6341             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
6342         end)
6343     lbr[id] = lbr[id] or {}
6344     table.insert(lbr[id],
6345         { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6346     }&%
6347 \endgroup}
6348 \endgroup
6349 \let\bbl@transfont@list@empty
6350 \def\bbl@settransfont{%
6351 \global\let\bbl@settransfont\relax % Execute only once
6352 \gdef\bbl@transfont{%
6353 \def\bbl@elt####1####2####3{%
6354 \bbl@ifblank{####3}%
6355 {\count@tw@}% Do nothing if no fonts
6356 {\count@z@
6357 \bbl@vforeach{####3}{%
6358 \def\bbl@tempd{#####1}%
6359 \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6360 \ifx\bbl@tempd\bbl@tempe
6361 \count@\@ne
6362 \else\ifx\bbl@tempd\bbl@transfam
6363 \count@\@ne
6364 \fi\fi}%
6365 \ifcase\count@
6366 \bbl@csarg\unsetattribute{ATR@###2@###1@###3}%
6367 \or
6368 \bbl@csarg\setattribute{ATR@###2@###1@###3}\@ne
6369 \fi}}%
6370 \bbl@transfont@list}%
6371 \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6372 \gdef\bbl@transfam{-unknown-}%
6373 \bbl@foreach\bbl@font@fams{%
6374 \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6375 \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6376 {\xdef\bbl@transfam{##1}}%
6377 {}}}
6378 \DeclareRobustCommand\enablelocaletransform[1]{%
6379 \bbl@ifunset{\bbl@ATR@#1@\languagename @}%
6380 {\bbl@error{transform-not-available}{#1}{}}%
6381 {\bbl@csarg\setattribute{ATR@#1@\languagename @}\@ne}}
6382 \DeclareRobustCommand\disablelocaletransform[1]{%
6383 \bbl@ifunset{\bbl@ATR@#1@\languagename @}%
6384 {\bbl@error{transform-not-available-b}{#1}{}}%
6385 {\bbl@csarg\unsetattribute{ATR@#1@\languagename @}}}
6386 \def\bbl@activateposthyphen{%
6387 \let\bbl@activateposthyphen\relax
6388 \directlua{
6389 require('babel-transforms.lua')
6390 Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6391 }}
6392 \def\bbl@activateprehyphen{%

```

```

6393 \let\bbl@activateprehyphen\relax
6394 \directlua{
6395     require('babel-transforms.lua')
6396     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6397 }}

```

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.

```

6398 \newcommand\localeprehyphenation[1]{%
6399 \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.

```

6400 \def\bbl@activate@preotf{%
6401 \let\bbl@activate@preotf\relax % only once
6402 \directlua{
6403     Babel = Babel or {}
6404     %
6405     function Babel.pre_otfload_v(head)
6406         if Babel.numbers and Babel.digits_mapped then
6407             head = Babel.numbers(head)
6408         end
6409         if Babel.bidi_enabled then
6410             head = Babel.bidi(head, false, dir)
6411         end
6412         return head
6413     end
6414     %
6415     function Babel.pre_otfload_h(head, gc, sz, pt, dir) %%% TODO
6416         if Babel.numbers and Babel.digits_mapped then
6417             head = Babel.numbers(head)
6418         end
6419         if Babel.bidi_enabled then
6420             head = Babel.bidi(head, false, dir)
6421         end
6422         return head
6423     end
6424     %
6425     luatexbase.add_to_callback('pre_linebreak_filter',
6426         Babel.pre_otfload_v,
6427         'Babel.pre_otfload_v',
6428         luatexbase.priority_in_callback('pre_linebreak_filter',
6429             'luaotfload.node_processor') or nil)
6430     %
6431     luatexbase.add_to_callback('hpack_filter',
6432         Babel.pre_otfload_h,
6433         'Babel.pre_otfload_h',
6434         luatexbase.priority_in_callback('hpack_filter',
6435             'luaotfload.node_processor') or nil)
6436 }}

```

The basic setup. The output is modified at a very low level to set the `\bodydir` to the `\pagedir`. Sadly, we have to deal with boxes in math with basic, so the `\bbl@mathboxdir` hack is activated every math with the package option `bidi=`. The hack for the PUA is no longer necessary with basic (24.8), but it's kept in `basic-r`.

```

6437 \breakafterdirmode=1
6438 \ifnum\bbl@bidimode>\@ne % Any bidi= except default (=1)

```

```

6439 \let\bbbl@beforeforeign\leavevmode
6440 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6441 \RequirePackage{luatexbase}
6442 \bbbl@activate@preotf
6443 \directlua{
6444   require('babel-data-bidi.lua')
6445   \ifcase\expandafter\@gobbletwo\the\bbbl@bidimode\or
6446     require('babel-bidi-basic.lua')
6447   \or
6448     require('babel-bidi-basic-r.lua')
6449     table.insert(Babel.ranges, {0xE000, 0xF8FF, 'on'})
6450     table.insert(Babel.ranges, {0xF000, 0xFFFFD, 'on'})
6451     table.insert(Babel.ranges, {0x10000, 0x10FFFD, 'on'})
6452   \fi}
6453 \newattribute\bbbl@attr@dir
6454 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbbl@attr@dir' }
6455 \bbbl@exp{\output{\bodydir\pagedir\the\output}}
6456 \fi
6457 \chardef\bbbl@thetextdir\z@
6458 \chardef\bbbl@thepardir\z@
6459 \def\bbbl@getluadir#1{%
6460   \directlua{
6461     if tex.#ldir == 'TLT' then
6462       tex.sprint('0')
6463     elseif tex.#ldir == 'TRT' then
6464       tex.sprint('1')
6465     end}}
6466 \def\bbbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6467   \ifcase#3\relax
6468     \ifcase\bbbl@getluadir{#1}\relax\else
6469       #2 TLT\relax
6470     \fi
6471   \else
6472     \ifcase\bbbl@getluadir{#1}\relax
6473       #2 TRT\relax
6474     \fi
6475   \fi}
6476 % ..00PPTT, with masks 0xC (par dir) and 0x3 (text dir)
6477 \def\bbbl@thedir{0}
6478 \def\bbbl@textdir#1{%
6479   \bbbl@setluadir{text}\textdir{#1}%
6480   \chardef\bbbl@thetextdir#1\relax
6481   \edef\bbbl@thedir{\the\numexpr\bbbl@thepardir*4+#1}%
6482   \setattribute\bbbl@attr@dir{\numexpr\bbbl@thepardir*4+#1}}
6483 \def\bbbl@pardir#1{% Used twice
6484   \bbbl@setluadir{par}\pardir{#1}%
6485   \chardef\bbbl@thepardir#1\relax}
6486 \def\bbbl@bodydir{\bbbl@setluadir{body}\bodydir}% Used once
6487 \def\bbbl@pagedir{\bbbl@setluadir{page}\pagedir}% Unused
6488 \def\bbbl@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.

6489 \ifnum\bbbl@bidimode>\z@ % Any bidi=
6490   \def\bbbl@insidemath{0}%
6491   \def\bbbl@everymath{\def\bbbl@insidemath{1}}
6492   \def\bbbl@everydisplay{\def\bbbl@insidemath{2}}
6493   \frozen@everymath\expandafter{%
6494     \expandafter\bbbl@everymath\the\frozen@everymath}
6495   \frozen@everydisplay\expandafter{%
6496     \expandafter\bbbl@everydisplay\the\frozen@everydisplay}
6497 \AtBeginDocument{
6498   \directlua{

```

```

6499     function Babel.math_box_dir(head)
6500         if not (token.get_macro('bbl@insidemath') == '0') then
6501             if Babel.hlist_has_bidi(head) then
6502                 local d = node.new(node.id'dir')
6503                 d.dir = '+TRT'
6504                 node.insert_before(head, node.has_glyph(head), d)
6505                 local inmath = false
6506                 for item in node.traverse(head) do
6507                     if item.id == 11 then
6508                         inmath = (item.subtype == 0)
6509                     elseif not inmath then
6510                         node.set_attribute(item,
6511                             Babel.attr_dir, token.get_macro('bbl@thedir'))
6512                     end
6513                 end
6514             end
6515         end
6516         return head
6517     end
6518     luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6519         "Babel.math_box_dir", 0)
6520     if Babel.unset_atdir then
6521         luatexbase.add_to_callback("pre_linebreak_filter", Babel.unset_atdir,
6522             "Babel.unset_atdir")
6523         luatexbase.add_to_callback("hpack_filter", Babel.unset_atdir,
6524             "Babel.unset_atdir")
6525     end
6526 }}%
6527 \fi

```

Experimental. Tentative name.

```

6528 \DeclareRobustCommand\localebox[1]{%
6529     {\def\bbl@insidemath{0}%
6530     \mbox{\foreignlanguage{\languagename}{#1}}}}

```

10.10 Layout

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

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

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

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

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

```

6531 \bbl@trace{Redefinitions for bidi layout}
6532 %
6533 <<{*More package options}>> ≡
6534 \chardef\bbl@eqnpos\z@
6535 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6536 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6537 <</More package options>>

```

```

6538 %
6539 \ifnum\bb@bidimode>\z@ % Any bidi=
6540 \matheqdirmode\@ne % A luatex primitive
6541 \let\bb@eqnodir\relax
6542 \def\bb@eqdel{()}
6543 \def\bb@eqnum{%
6544   {\normalfont\normalcolor
6545     \expandafter\@firstoftwo\bb@eqdel
6546     \theequation
6547     \expandafter\@secondoftwo\bb@eqdel}}
6548 \def\bb@puteqno#1{\eqno\hbox{#1}}
6549 \def\bb@putleqno#1{\leqno\hbox{#1}}
6550 \def\bb@eqno@flip#1{%
6551   \ifdim\predisplaysize=-\maxdimen
6552     \eqno
6553     \hb@xt@.01pt{%
6554       \hb@xt@\displaywidth{\hss{#1}\glet\bb@upset\@currentlabel}}\hss}%
6555   \else
6556     \leqno\hbox{#1}\glet\bb@upset\@currentlabel}%
6557   \fi
6558   \bb@exp{\def\@currentlabel{\bb@upset}}}
6559 \def\bb@leqno@flip#1{%
6560   \ifdim\predisplaysize=-\maxdimen
6561     \leqno
6562     \hb@xt@.01pt{%
6563       \hss\hb@xt@\displaywidth{\#1}\glet\bb@upset\@currentlabel}\hss}%
6564   \else
6565     \eqno\hbox{#1}\glet\bb@upset\@currentlabel}%
6566   \fi
6567   \bb@exp{\def\@currentlabel{\bb@upset}}}
6568 \AtBeginDocument{%
6569   \ifx\bb@noamsmath\relax\else
6570   \ifx\maketag@@@\@undefined % Normal equation, eqnarray
6571     \AddToHook{env/equation/begin}{%
6572       \ifnum\bb@thetextdir>\z@
6573         \def\bb@mathboxdir{\def\bb@insidemath{1}}%
6574         \let\@eqnum\bb@eqnum
6575         \edef\bb@eqnodir{\noexpand\bb@textdir\the\bb@thetextdir}%
6576         \chardef\bb@thetextdir\z@
6577         \bb@add\normalfont{\bb@eqnodir}%
6578         \ifcase\bb@eqnpos
6579           \let\bb@puteqno\bb@eqno@flip
6580         \or
6581           \let\bb@puteqno\bb@leqno@flip
6582         \fi
6583       \fi}%
6584     \ifnum\bb@eqnpos=\tw@\else
6585     \def\endequation{\bb@puteqno{\@eqnum}$$\@ignoretrue}%
6586     \fi
6587     \AddToHook{env/eqnarray/begin}{%
6588       \ifnum\bb@thetextdir>\z@
6589         \def\bb@mathboxdir{\def\bb@insidemath{1}}%
6590         \edef\bb@eqnodir{\noexpand\bb@textdir\the\bb@thetextdir}%
6591         \chardef\bb@thetextdir\z@
6592         \bb@add\normalfont{\bb@eqnodir}%
6593         \ifnum\bb@eqnpos=\@ne
6594           \def\@eqnum{%
6595             \setbox\z@\hbox{\bb@eqnum}%
6596             \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6597           \else
6598             \let\@eqnum\bb@eqnum
6599           \fi
6600         \fi}

```

```

6601 % Hack. YA luatex bug?:
6602 \expandafter\bb@l@sreplace\csname] \endcsname{${$}\eqno\kern.001pt${$}}%
6603 \else % amstex
6604 \bb@l@exp{% Hack to hide maybe undefined conditionals:
6605 \chardef\bb@l@eqnpos=0%
6606 \<iftagsleft@>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6607 \ifnum\bb@l@eqnpos=\@ne
6608 \let\bb@l@ams@lap\hbox
6609 \else
6610 \let\bb@l@ams@lap\llap
6611 \fi
6612 \ExplSyntaxOn % Required by \bb@l@sreplace with \intertext@
6613 \bb@l@sreplace\intertext@{\normalbaselines}%
6614 {\normalbaselines
6615 \ifx\bb@l@eqnodir\relax\else\bb@l@pardir\@ne\bb@l@eqnodir\fi}%
6616 \ExplSyntaxOff
6617 \def\bb@l@ams@tagbox#1#2{#1{\bb@l@eqnodir#2}}% #1=hbox|lap|flip
6618 \ifx\bb@l@ams@lap\hbox % leqno
6619 \def\bb@l@ams@flip#1{%
6620 \hbox to 0.01pt{\hss\hbox to\displaywidth{#1}\hss}}%
6621 \else % eqno
6622 \def\bb@l@ams@flip#1{%
6623 \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}}\hss}}%
6624 \fi
6625 \def\bb@l@ams@preset#1{%
6626 \def\bb@l@mathboxdir{\def\bb@l@insidemath{1}}%
6627 \ifnum\bb@l@thetextdir>\z@
6628 \edef\bb@l@eqnodir{\noexpand\bb@l@textdir{\the\bb@l@thetextdir}}%
6629 \bb@l@sreplace\textdef@{\hbox}{\bb@l@ams@tagbox\hbox}%
6630 \bb@l@sreplace\maketag@@@{\hbox}{\bb@l@ams@tagbox#1}%
6631 \fi}%
6632 \ifnum\bb@l@eqnpos=\tw@ \else
6633 \def\bb@l@ams@equation{%
6634 \def\bb@l@mathboxdir{\def\bb@l@insidemath{1}}%
6635 \ifnum\bb@l@thetextdir>\z@
6636 \edef\bb@l@eqnodir{\noexpand\bb@l@textdir{\the\bb@l@thetextdir}}%
6637 \chardef\bb@l@thetextdir\z@
6638 \bb@l@add\normalfont{\bb@l@eqnodir}%
6639 \ifcase\bb@l@eqnpos
6640 \def\veqno##1##2{\bb@l@eqno@flip{##1##2}}%
6641 \or
6642 \def\veqno##1##2{\bb@l@leqno@flip{##1##2}}%
6643 \fi
6644 \fi}%
6645 \AddToHook{env/equation/begin}{\bb@l@ams@equation}%
6646 \AddToHook{env/equation*/begin}{\bb@l@ams@equation}%
6647 \fi
6648 \AddToHook{env/cases/begin}{\bb@l@ams@preset\bb@l@ams@lap}%
6649 \AddToHook{env/multline/begin}{\bb@l@ams@preset\hbox}%
6650 \AddToHook{env/gather/begin}{\bb@l@ams@preset\bb@l@ams@lap}%
6651 \AddToHook{env/gather*/begin}{\bb@l@ams@preset\bb@l@ams@lap}%
6652 \AddToHook{env/align/begin}{\bb@l@ams@preset\bb@l@ams@lap}%
6653 \AddToHook{env/align*/begin}{\bb@l@ams@preset\bb@l@ams@lap}%
6654 \AddToHook{env/alignat/begin}{\bb@l@ams@preset\bb@l@ams@lap}%
6655 \AddToHook{env/alignat*/begin}{\bb@l@ams@preset\bb@l@ams@lap}%
6656 \AddToHook{env/eqnalign/begin}{\bb@l@ams@preset\hbox}%
6657 % Hackish, for proper alignment. Don't ask me why it works!:)
6658 \bb@l@exp{% Avoid a 'visible' conditional
6659 \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\tag*}{\<fi>}%
6660 \\\AddToHook{env/alignat*/end}{\<iftag@>\<else>\\tag*}{\<fi>}}%
6661 \AddToHook{env/flalign/begin}{\bb@l@ams@preset\hbox}%
6662 \AddToHook{env/split/before}{%
6663 \def\bb@l@mathboxdir{\def\bb@l@insidemath{1}}%

```

```

6664     \ifnum\bb1@thetextdir>\z@
6665     \bb1@ifsamestring\@currentvir{equation}%
6666     {\ifx\bb1@ams@lap\hbox % leqno
6667     \def\bb1@ams@flip#1{%
6668     \hbox to 0.01pt{\hbox to\displaywidth{#1}\hss}\hss}%
6669     \else
6670     \def\bb1@ams@flip#1{%
6671     \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}%
6672     \fi}%
6673     }%
6674     \fi}%
6675     \fi\fi}
6676 \fi
6677 \def\bb1@provide@extra#1{%
6678 % == Counters: mapdigits ==
6679 % Native digits
6680 \ifx\bb1@KVP@mapdigits\@nnil\else
6681 \bb1@ifunset{bb1@dgnat@\languagename}{}%
6682 {\RequirePackage{luatexbase}%
6683 \bb1@activate@preotf
6684 \directlua{
6685     Babel = Babel or {} %%% -> presets in luababel
6686     Babel.digits_mapped = true
6687     Babel.digits = Babel.digits or {}
6688     Babel.digits[\the\localeid] =
6689     table.pack(string.utfvalue('\bb1@cl{dgnat}'))
6690     if not Babel.numbers then
6691     function Babel.numbers(head)
6692     local LOCALE = Babel.attr_locale
6693     local GLYPH = node.id'glyph'
6694     local inmath = false
6695     for item in node.traverse(head) do
6696     if not inmath and item.id == GLYPH then
6697     local temp = node.get_attribute(item, LOCALE)
6698     if Babel.digits[temp] then
6699     local chr = item.char
6700     if chr > 47 and chr < 58 then
6701     item.char = Babel.digits[temp][chr-47]
6702     end
6703     end
6704     elseif item.id == node.id'math' then
6705     inmath = (item.subtype == 0)
6706     end
6707     end
6708     return head
6709     end
6710     end
6711     }}%
6712 \fi
6713 % == transforms ==
6714 \ifx\bb1@KVP@transforms\@nnil\else
6715 \def\bb1@elt##1##2##3{%
6716 \in@{${transforms.}{##1}%
6717 \ifin@
6718 \def\bb1@tempa{##1}%
6719 \bb1@replace\bb1@tempa{transforms.}{}%
6720 \bb1@carg\bb1@transforms{babel\bb1@tempa}{##2}{##3}%
6721 \fi}%
6722 \bb1@exp{%
6723 \\bb1@ifblank{\bb1@cl{dgnat}}}%
6724 {\let\\bb1@tempa\relax}%
6725 {\def\\bb1@tempa{%
6726 \\bb1@elt{transforms.prehyphenation}%

```



```

6727         {digits.native.1.0}{([0-9])}%
6728         \\bbl@elt{transforms.prehyphenation}%
6729         {digits.native.1.1}{string={1|string|0123456789|string|bbl@cl{dgnat}}}}}%
6730     \ifx\bbl@tempa\relax\else
6731         \toks@\expandafter\expandafter\expandafter{%
6732             \csname bbl@inidata@\languagename\endcsname}%
6733         \bbl@csarg\edef{inidata@\languagename}{%
6734             \unexpanded\expandafter{\bbl@tempa}%
6735             \the\toks@}%
6736     \fi
6737     \csname bbl@inidata@\languagename\endcsname
6738     \bbl@release@transforms\relax % \relax closes the last item.
6739 \fi}

```

Start tabular here:

```

6740 \def\localerestoredirs{%
6741     \ifcase\bbl@thetextdir
6742         \ifnum\textdirection=\z@\else\textdir TLT\fi
6743     \else
6744         \ifnum\textdirection=\@ne\else\textdir TRT\fi
6745     \fi
6746     \ifcase\bbl@thepardir
6747         \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6748     \else
6749         \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6750     \fi}
6751 \IfBabelLayout{tabular}%
6752     {\chardef\bbl@tabular@mode\tw@}% All RTL
6753     {\IfBabelLayout{notabular}%
6754         {\chardef\bbl@tabular@mode\z@}%
6755         {\chardef\bbl@tabular@mode\@ne}}% Mixed, with LTR cols
6756 \ifnum\bbl@bidimode>\@ne % Any lua bidi= except default=1
6757 % Redefine: vrules mess up dirs. TODO: why?
6758 \def\@arstrut{\relax\copy\@arstrutbox}%
6759 \ifcase\bbl@tabular@mode\or % 1 = Mixed - default
6760     \let\bbl@parabefore\relax
6761     \AddToHook{para/before}{\bbl@parabefore}
6762     \AtBeginDocument{%
6763         \bbl@replace\@tabular{\$}{\$}%
6764         \def\bbl@insidemath{0}%
6765         \def\bbl@parabefore{\localerestoredirs}}%
6766     \ifnum\bbl@tabular@mode=\@ne
6767         \bbl@ifunset{\@tabclassz}{}%
6768         \bbl@exp{% Hide conditionals
6769             \\bbl@sreplace\\ \@tabclassz
6770             {\<ifcase>\\ \@chnum}%
6771             {\localerestoredirs\<ifcase>\\ \@chnum}}}%
6772     \@ifpackageloaded{colortbl}%
6773         {\bbl@sreplace\@classz
6774             {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6775     {\@ifpackageloaded{array}%
6776         {\bbl@exp{% Hide conditionals
6777             \\bbl@sreplace\\ \@classz
6778             {\<ifcase>\\ \@chnum}%
6779             {\bgroup\\ \localerestoredirs\<ifcase>\\ \@chnum}%
6780             \\bbl@sreplace\\ \@classz
6781             {\do@row@strut\<fi>}{\do@row@strut\<fi>\egroup}}}%
6782         {}}%
6783     \fi}%
6784 \or % 2 = All RTL - tabular
6785     \let\bbl@parabefore\relax
6786     \AddToHook{para/before}{\bbl@parabefore}%
6787     \AtBeginDocument{%

```

```

6788 \ifpackageloaded{colortbl}%
6789   {\bbl@replace@tabular{$}{$%
6790     \def\bbl@insidemath{0}%
6791     \def\bbl@parabefore{\localerestoredirs}}%
6792     \bbl@sreplace@classz
6793     {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6794     }%
6795 \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.

```

6796 \AtBeginDocument{%
6797   \ifpackageloaded{multicol}%
6798     {\toks@\expandafter{\multi@column@out}%
6799     \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
6800     }%
6801   \ifpackageloaded{paracol}%
6802     {\edef\pcol@output{%
6803       \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
6804     }%
6805 \fi
6806 \ifx\bbl@opt@layout@annil\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`.

```

6807 \ifnum\bbl@bidimode>\z@ % Any bidi=
6808   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6809     \bbl@exp{%
6810       \mathdir\the\bodydir
6811       #1% Once entered in math, set boxes to restore values
6812       \def\\bbl@insidemath{0}%
6813       \<ifmmode>%
6814         \everyvbox{%
6815           \the\everyvbox
6816           \bodydir\the\bodydir
6817           \mathdir\the\mathdir
6818           \everyhbox{\the\everyhbox}%
6819           \everyvbox{\the\everyvbox}}%
6820         \everyhbox{%
6821           \the\everyhbox
6822           \bodydir\the\bodydir
6823           \mathdir\the\mathdir
6824           \everyhbox{\the\everyhbox}%
6825           \everyvbox{\the\everyvbox}}%
6826         \<fi>}}%
6827   \def\@hangfrom#1{%
6828     \setbox\@tempboxa\hbox{#1}%
6829     \hangindent\wd\@tempboxa
6830     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6831       \shapemode\@ne
6832       \fi
6833     \noindent\box\@tempboxa}
6834 \fi
6835 \IfBabelLayout{tabular}
6836   {\let\bbl@OL@tabular\@tabular
6837   \bbl@replace@tabular{$}{\bbl@nextfake$}%
6838   \let\bbl@NL@tabular\@tabular
6839   \AtBeginDocument{%
6840     \ifx\bbl@NL@tabular\@tabular\else
6841       \bbl@exp{\in{\bbl@nextfake}{\@tabular}}%
6842     \ifin\else

```

```

6843     \bbl@replace\@tabular{$}\bbl@nextfake$}%
6844     \fi
6845     \let\bbl@NL@\@tabular\@tabular
6846     \fi}}
6847   {}
6848 \IfBabelLayout{lists}
6849   {\let\bbl@OL@list\list
6850    \bbl@sreplace\list{\parshape}\bbl@listparshape}%
6851    \let\bbl@NL@list\list
6852    \def\bbl@listparshape#1#2#3{%
6853     \parshape #1 #2 #3 %
6854     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6855       \shapemode\tw@
6856     \fi}}
6857   {}
6858 \IfBabelLayout{graphics}
6859   {\let\bbl@pictresetdir\relax
6860    \def\bbl@pictsetdir#1{%
6861     \ifcase\bbl@thetextdir
6862     \let\bbl@pictresetdir\relax
6863     \else
6864     \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6865     \or\textdir TLT
6866     \else\bodydir TLT \textdir TLT
6867     \fi
6868     % \(\text|par)dir required in pgf:
6869     \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6870     \fi}%
6871 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6872 \directlua{
6873   Babel.get_picture_dir = true
6874   Babel.picture_has_bidi = 0
6875   %
6876   function Babel.picture_dir (head)
6877     if not Babel.get_picture_dir then return head end
6878     if Babel.hlist_has_bidi(head) then
6879       Babel.picture_has_bidi = 1
6880     end
6881     return head
6882   end
6883   luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6884     "Babel.picture_dir")
6885 }%
6886 \AtBeginDocument{%
6887   \def\LS@rot{%
6888     \setbox\@outputbox\vbox{%
6889     \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
6890   \long\def\put(#1,#2)#3{%
6891     \@killglue
6892     % Try:
6893     \ifx\bbl@pictresetdir\relax
6894       \def\bbl@tempc{0}%
6895     \else
6896       \directlua{
6897         Babel.get_picture_dir = true
6898         Babel.picture_has_bidi = 0
6899       }%
6900     \setbox\z@\hb@xt@\z@{%
6901     \@defaultunitsset\@tempdimc{#1}\unitlength
6902     \kern\@tempdimc
6903     #3\hss}% TODO: #3 executed twice (below). That's bad.
6904     \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6905     \fi

```

```

6906 % Do:
6907 \@defaultunitsset\@tempdimc{#2}\unitlength
6908 \raise\@tempdimc\hbext\z@{\%
6909 \@defaultunitsset\@tempdimc{#1}\unitlength
6910 \kern\@tempdimc
6911 {\ifnum\bbbl@tempc>\z@\bbbl@pictresetdir\fi#3}\hss}%
6912 \ignorespaces}%
6913 \MakeRobust\put}%
6914 \AtBeginDocument
6915 {\AddToHook{cmd/diagbox@pict/before}{\let\bbbl@pictsetdir\@gobble}%
6916 \ifx\pgfpicture\undefined\else % TODO. Allow deactivate?
6917 \AddToHook{env/pgfpicture/begin}{\bbbl@pictsetdir\@ne}%
6918 \bbbl@add\pgfinterruptpicture{\bbbl@pictresetdir}%
6919 \bbbl@add\pgfsys@beginpicture{\bbbl@pictsetdir\z@}%
6920 \fi
6921 \ifx\tikzpicture\undefined\else
6922 \AddToHook{env/tikzpicture/begin}{\bbbl@pictsetdir\tw@}%
6923 \bbbl@add\tikz@atbegin@node{\bbbl@pictresetdir}%
6924 \bbbl@replace\tikz{\begingroup}{\begingroup\bbbl@pictsetdir\tw@}%
6925 \fi
6926 \ifx\tcolorbox\undefined\else
6927 \def\tcb@drawing@env@begin{%
6928 \csname tcb@before@\tcb@split@state\endcsname
6929 \bbbl@pictsetdir\tw@
6930 \begin{\kvtcb@graphenv}%
6931 \tcb@bbdraw
6932 \tcb@apply@graph@patches}%
6933 \def\tcb@drawing@env@end{%
6934 \end{\kvtcb@graphenv}%
6935 \bbbl@pictresetdir
6936 \csname tcb@after@\tcb@split@state\endcsname}%
6937 \fi
6938 }}
6939 {}

```

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.

```

6940 \IfBabelLayout{counters*}%
6941 {\bbbl@add\bbbl@opt@layout{.counters.}%
6942 \directlua{
6943 \luatexbase.add_to_callback("process_output_buffer",
6944 Babel.discard_sublr , "Babel.discard_sublr") }%
6945 }}
6946 \IfBabelLayout{counters}%
6947 {\let\bbbl@0L@@textsuperscript\@textsuperscript
6948 \bbbl@replace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6949 \let\bbbl@latinarabic=\@arabic
6950 \let\bbbl@0L@@arabic\@arabic
6951 \def\@arabic#1{\babelsublr{\bbbl@latinarabic#1}}%
6952 \@ifpackagewith{babel}{bidi=default}%
6953 {\let\bbbl@asciroman=\@roman
6954 \let\bbbl@0L@@roman\@roman
6955 \def\@roman#1{\babelsublr{\ensureascii{\bbbl@asciroman#1}}}%
6956 \let\bbbl@asciiRoman=\@Roman
6957 \let\bbbl@0L@@roman\@Roman
6958 \def\@Roman#1{\babelsublr{\ensureascii{\bbbl@asciiRoman#1}}}%
6959 \let\bbbl@0L@labelenumii\labelenumii
6960 \def\labelenumii()\theenumii}%
6961 \let\bbbl@0L@p@enumiii\p@enumiii
6962 \def\p@enumiii{\p@enumii}\theenumii(){}{}{}}
6963 <@Footnote changes@>
6964 \IfBabelLayout{footnotes}%

```

```

6965 {\let\bbl@0L@footnote\footnote
6966 \BabelFootnote\footnote\languagename{}}}%
6967 \BabelFootnote\localfootnote\languagename{}}}%
6968 \BabelFootnote\mainfootnote{}}{}}
6969 {}

```

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.

```

6970 \IfBabelLayout{extras}%
6971 {\bbl@ncarg\let\bbl@0L@underline{underline }%
6972 \bbl@carg\bbl@sreplace{underline }%
6973 {\$@@underline}{\bgroup\bbl@nextfake$@@underline}%
6974 \bbl@carg\bbl@sreplace{underline }%
6975 {\m@th$}{\m@th$\egroup}%
6976 \let\bbl@0L@LaTeXe\LaTeXe
6977 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6978 \if b\expandafter\@car\f@series\@nil\boldmath\fi
6979 \babelsublr}%
6980 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}
6981 {}
6982 </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.

```

6983 <:*transforms>
6984 Babel.linebreaking.replacements = {}
6985 Babel.linebreaking.replacements[0] = {} -- pre
6986 Babel.linebreaking.replacements[1] = {} -- post
6987
6988 function Babel.tovalue(v)
6989   if type(v) == 'string' then
6990     return loadstring('return ' .. v)()
6991   else
6992     return v
6993   end
6994 end
6995
6996 -- Discretionaries contain strings as nodes
6997 function Babel.str_to_nodes(fn, matches, base)
6998   local n, head, last
6999   if fn == nil then return nil end
7000   for s in string.utfvalues(fn(matches)) do
7001     if base.id == 7 then
7002       base = base.replace
7003     end
7004     n = node.copy(base)
7005     n.char = s
7006     if not head then
7007       head = n
7008     else
7009       last.next = n
7010     end

```

```

7011     last = n
7012 end
7013 return head
7014 end
7015
7016 Babel.fetch_subtext = {}
7017
7018 Babel.ignore_pre_char = function(node)
7019     return (node.lang == Babel.nohyphenation)
7020 end
7021
7022 -- Merging both functions doesn't seem feasible, because there are too
7023 -- many differences.
7024 Babel.fetch_subtext[0] = function(head)
7025     local word_string = ''
7026     local word_nodes = {}
7027     local lang
7028     local item = head
7029     local inmath = false
7030
7031     while item do
7032
7033         if item.id == 11 then
7034             inmath = (item.subtype == 0)
7035         end
7036
7037         if inmath then
7038             -- pass
7039
7040         elseif item.id == 29 then
7041             local locale = node.get_attribute(item, Babel.attr_locale)
7042
7043             if lang == locale or lang == nil then
7044                 lang = lang or locale
7045                 if Babel.ignore_pre_char(item) then
7046                     word_string = word_string .. Babel.us_char
7047                 else
7048                     word_string = word_string .. unicode.utf8.char(item.char)
7049                 end
7050                 word_nodes[#word_nodes+1] = item
7051             else
7052                 break
7053             end
7054
7055         elseif item.id == 12 and item.subtype == 13 then
7056             word_string = word_string .. ' '
7057             word_nodes[#word_nodes+1] = item
7058
7059             -- Ignore leading unrecognized nodes, too.
7060         elseif word_string ~= '' then
7061             word_string = word_string .. Babel.us_char
7062             word_nodes[#word_nodes+1] = item -- Will be ignored
7063         end
7064
7065         item = item.next
7066     end
7067
7068     -- Here and above we remove some trailing chars but not the
7069     -- corresponding nodes. But they aren't accessed.
7070     if word_string:sub(-1) == ' ' then
7071         word_string = word_string:sub(1,-2)
7072     end
7073     word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')

```

```

7074 return word_string, word_nodes, item, lang
7075 end
7076
7077 Babel.fetch_subtext[1] = function(head)
7078   local word_string = ''
7079   local word_nodes = {}
7080   local lang
7081   local item = head
7082   local inmath = false
7083
7084   while item do
7085
7086     if item.id == 11 then
7087       inmath = (item.subtype == 0)
7088     end
7089
7090     if inmath then
7091       -- pass
7092
7093     elseif item.id == 29 then
7094       if item.lang == lang or lang == nil then
7095         if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
7096           lang = lang or item.lang
7097           word_string = word_string .. unicode.utf8.char(item.char)
7098           word_nodes[#word_nodes+1] = item
7099         end
7100       else
7101         break
7102       end
7103
7104     elseif item.id == 7 and item.subtype == 2 then
7105       word_string = word_string .. '='
7106       word_nodes[#word_nodes+1] = item
7107
7108     elseif item.id == 7 and item.subtype == 3 then
7109       word_string = word_string .. '|'
7110       word_nodes[#word_nodes+1] = item
7111
7112     -- (1) Go to next word if nothing was found, and (2) implicitly
7113     -- remove leading USs.
7114     elseif word_string == '' then
7115       -- pass
7116
7117     -- This is the responsible for splitting by words.
7118     elseif (item.id == 12 and item.subtype == 13) then
7119       break
7120
7121     else
7122       word_string = word_string .. Babel.us_char
7123       word_nodes[#word_nodes+1] = item -- Will be ignored
7124     end
7125
7126     item = item.next
7127   end
7128
7129   word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7130   return word_string, word_nodes, item, lang
7131 end
7132
7133 function Babel.pre_hyphenate_replace(head)
7134   Babel.hyphenate_replace(head, 0)
7135 end
7136

```

```

7137 function Babel.post_hyphenate_replace(head)
7138   Babel.hyphenate_replace(head, 1)
7139 end
7140
7141 Babel.us_char = string.char(31)
7142
7143 function Babel.hyphenate_replace(head, mode)
7144   local u = unicode.utf8
7145   local lbkr = Babel.linebreaking.replacements[mode]
7146
7147   local word_head = head
7148
7149   while true do -- for each subtext block
7150
7151     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
7152
7153     if Babel.debug then
7154       print()
7155       print((mode == 0) and '@@@@<' or '@@@@>', w)
7156     end
7157
7158     if nw == nil and w == '' then break end
7159
7160     if not lang then goto next end
7161     if not lbkr[lang] then goto next end
7162
7163     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
7164     -- loops are nested.
7165     for k=1, #lbkr[lang] do
7166       local p = lbkr[lang][k].pattern
7167       local r = lbkr[lang][k].replace
7168       local attr = lbkr[lang][k].attr or -1
7169
7170       if Babel.debug then
7171         print('*****', p, mode)
7172       end
7173
7174       -- This variable is set in some cases below to the first *byte*
7175       -- after the match, either as found by u.match (faster) or the
7176       -- computed position based on sc if w has changed.
7177       local last_match = 0
7178       local step = 0
7179
7180       -- For every match.
7181       while true do
7182         if Babel.debug then
7183           print('====')
7184         end
7185         local new -- used when inserting and removing nodes
7186         local dummy_node -- used by after
7187
7188         local matches = { u.match(w, p, last_match) }
7189
7190         if #matches < 2 then break end
7191
7192         -- Get and remove empty captures (with ())'s, which return a
7193         -- number with the position), and keep actual captures
7194         -- (from (...)), if any, in matches.
7195         local first = table.remove(matches, 1)
7196         local last = table.remove(matches, #matches)
7197         -- Non re-fetched substrings may contain \31, which separates
7198         -- substrings.
7199         if string.find(w:sub(first, last-1), Babel.us_char) then break end

```



```

7200
7201     local save_last = last -- with A()BC()D, points to D
7202
7203     -- Fix offsets, from bytes to unicode. Explained above.
7204     first = u.len(w:sub(1, first-1)) + 1
7205     last = u.len(w:sub(1, last-1)) -- now last points to C
7206
7207     -- This loop stores in a small table the nodes
7208     -- corresponding to the pattern. Used by 'data' to provide a
7209     -- predictable behavior with 'insert' (w_nodes is modified on
7210     -- the fly), and also access to 'remove'd nodes.
7211     local sc = first-1          -- Used below, too
7212     local data_nodes = {}
7213
7214     local enabled = true
7215     for q = 1, last-first+1 do
7216         data_nodes[q] = w_nodes[sc+q]
7217         if enabled
7218             and attr > -1
7219             and not node.has_attribute(data_nodes[q], attr)
7220         then
7221             enabled = false
7222         end
7223     end
7224
7225     -- This loop traverses the matched substring and takes the
7226     -- corresponding action stored in the replacement list.
7227     -- sc = the position in substr nodes / string
7228     -- rc = the replacement table index
7229     local rc = 0
7230
7231     ----- TODO. dummy_node?
7232     while rc < last-first+1 or dummy_node do -- for each replacement
7233         if Babel.debug then
7234             print('.....', rc + 1)
7235         end
7236         sc = sc + 1
7237         rc = rc + 1
7238
7239         if Babel.debug then
7240             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7241             local ss = ''
7242             for itt in node.traverse(head) do
7243                 if itt.id == 29 then
7244                     ss = ss .. unicode.utf8.char(itt.char)
7245                 else
7246                     ss = ss .. '{' .. itt.id .. '}'
7247                 end
7248             end
7249             print('*****', ss)
7250         end
7251     end
7252
7253     local crep = r[rc]
7254     local item = w_nodes[sc]
7255     local item_base = item
7256     local placeholder = Babel.us_char
7257     local d
7258
7259     if crep and crep.data then
7260         item_base = data_nodes[crep.data]
7261     end
7262

```

```

7263     if crep then
7264         step = crep.step or step
7265     end
7266
7267     if crep and crep.after then
7268         crep.insert = true
7269         if dummy_node then
7270             item = dummy_node
7271         else -- TODO. if there is a node after?
7272             d = node.copy(item_base)
7273             head, item = node.insert_after(head, item, d)
7274             dummy_node = item
7275         end
7276     end
7277
7278     if crep and not crep.after and dummy_node then
7279         node.remove(head, dummy_node)
7280         dummy_node = nil
7281     end
7282
7283     if (not enabled) or (crep and next(crep) == nil) then -- = {}
7284         if step == 0 then
7285             last_match = save_last -- Optimization
7286         else
7287             last_match = utf8.offset(w, sc+step)
7288         end
7289         goto next
7290
7291     elseif crep == nil or crep.remove then
7292         node.remove(head, item)
7293         table.remove(w_nodes, sc)
7294         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7295         sc = sc - 1 -- Nothing has been inserted.
7296         last_match = utf8.offset(w, sc+1+step)
7297         goto next
7298
7299     elseif crep and crep.kashida then -- Experimental
7300         node.set_attribute(item,
7301             Babel.attr_kashida,
7302             crep.kashida)
7303         last_match = utf8.offset(w, sc+1+step)
7304         goto next
7305
7306     elseif crep and crep.string then
7307         local str = crep.string(matches)
7308         if str == '' then -- Gather with nil
7309             node.remove(head, item)
7310             table.remove(w_nodes, sc)
7311             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7312             sc = sc - 1 -- Nothing has been inserted.
7313         else
7314             local loop_first = true
7315             for s in string.utfvalues(str) do
7316                 d = node.copy(item_base)
7317                 d.char = s
7318                 if loop_first then
7319                     loop_first = false
7320                     head, new = node.insert_before(head, item, d)
7321                     if sc == 1 then
7322                         word_head = head
7323                     end
7324                     w_nodes[sc] = d
7325                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)

```

```

7326         else
7327             sc = sc + 1
7328             head, new = node.insert_before(head, item, d)
7329             table.insert(w_nodes, sc, new)
7330             w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7331         end
7332         if Babel.debug then
7333             print('.....', 'str')
7334             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7335         end
7336         end -- for
7337         node.remove(head, item)
7338     end -- if ''
7339     last_match = utf8.offset(w, sc+1+step)
7340     goto next
7341
7342 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7343     d = node.new(7, 3) -- (disc, regular)
7344     d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7345     d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7346     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7347     d.attr = item_base.attr
7348     if crep.pre == nil then -- TeXbook p96
7349         d.penalty = crep.penalty or tex.hyphenpenalty
7350     else
7351         d.penalty = crep.penalty or tex.exhyphenpenalty
7352     end
7353     placeholder = '|'
7354     head, new = node.insert_before(head, item, d)
7355
7356 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7357     -- ERROR
7358
7359 elseif crep and crep.penalty then
7360     d = node.new(14, 0) -- (penalty, userpenalty)
7361     d.attr = item_base.attr
7362     d.penalty = crep.penalty
7363     head, new = node.insert_before(head, item, d)
7364
7365 elseif crep and crep.space then
7366     -- 655360 = 10 pt = 10 * 65536 sp
7367     d = node.new(12, 13) -- (glue, spaceskip)
7368     local quad = font.getfont(item_base.font).size or 655360
7369     node.setglue(d, crep.space[1] * quad,
7370                 crep.space[2] * quad,
7371                 crep.space[3] * quad)
7372     if mode == 0 then
7373         placeholder = ' '
7374     end
7375     head, new = node.insert_before(head, item, d)
7376
7377 elseif crep and crep.norule then
7378     -- 655360 = 10 pt = 10 * 65536 sp
7379     d = node.new(2, 3) -- (rule, empty) = \no*rule
7380     local quad = font.getfont(item_base.font).size or 655360
7381     d.width = crep.norule[1] * quad
7382     d.height = crep.norule[2] * quad
7383     d.depth = crep.norule[3] * quad
7384     head, new = node.insert_before(head, item, d)
7385
7386 elseif crep and crep.spacefactor then
7387     d = node.new(12, 13) -- (glue, spaceskip)
7388     local base_font = font.getfont(item_base.font)

```

```

7389         node.setglue(d,
7390             crep.spacefactor[1] * base_font.parameters['space'],
7391             crep.spacefactor[2] * base_font.parameters['space_stretch'],
7392             crep.spacefactor[3] * base_font.parameters['space_shrink'])
7393     if mode == 0 then
7394         placeholder = ' '
7395     end
7396     head, new = node.insert_before(head, item, d)
7397
7398     elseif mode == 0 and crep and crep.space then
7399         -- ERROR
7400
7401     elseif crep and crep.kern then
7402         d = node.new(13, 1) -- (kern, user)
7403         local quad = font.getfont(item_base.font).size or 655360
7404         d.attr = item_base.attr
7405         d.kern = crep.kern * quad
7406         head, new = node.insert_before(head, item, d)
7407
7408     elseif crep and crep.node then
7409         d = node.new(crep.node[1], crep.node[2])
7410         d.attr = item_base.attr
7411         head, new = node.insert_before(head, item, d)
7412
7413     end -- ie replacement cases
7414
7415     -- Shared by disc, space(factor), kern, node and penalty.
7416     if sc == 1 then
7417         word_head = head
7418     end
7419     if crep.insert then
7420         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7421         table.insert(w_nodes, sc, new)
7422         last = last + 1
7423     else
7424         w_nodes[sc] = d
7425         node.remove(head, item)
7426         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7427     end
7428
7429     last_match = utf8.offset(w, sc+1+step)
7430
7431     ::next::
7432
7433     end -- for each replacement
7434
7435     if Babel.debug then
7436         print('.....', '/')
7437         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7438     end
7439
7440     if dummy_node then
7441         node.remove(head, dummy_node)
7442         dummy_node = nil
7443     end
7444
7445     end -- for match
7446
7447     end -- for patterns
7448
7449     ::next::
7450     word_head = nw
7451     end -- for substring

```

```

7452 return head
7453 end
7454
7455 -- This table stores capture maps, numbered consecutively
7456 Babel.capture_maps = {}
7457
7458 -- The following functions belong to the next macro
7459 function Babel.capture_func(key, cap)
7460 local ret = "[[" .. cap:gsub('{{[0-9]}}', ")]..m[%1]..[" .. "]"
7461 local cnt
7462 local u = unicode.utf8
7463 ret, cnt = ret:gsub('{{[0-9]}|([^\]|+)|(-)}', Babel.capture_func_map)
7464 if cnt == 0 then
7465     ret = u.gsub(ret, '{{(%x%x%x%x+)}}',
7466         function (n)
7467             return u.char(tonumber(n, 16))
7468         end)
7469 end
7470 ret = ret:gsub("%[%[%]]%.%", '')
7471 ret = ret:gsub("%.%.%[%[%]]%", '')
7472 return key .. [[=function(m) return ]] .. ret .. [[ end]]
7473 end
7474
7475 function Babel.capt_map(from, mapno)
7476 return Babel.capture_maps[mapno][from] or from
7477 end
7478
7479 -- Handle the {n|abc|ABC} syntax in captures
7480 function Babel.capture_func_map(capno, from, to)
7481 local u = unicode.utf8
7482 from = u.gsub(from, '{{(%x%x%x%x+)}}',
7483     function (n)
7484         return u.char(tonumber(n, 16))
7485     end)
7486 to = u.gsub(to, '{{(%x%x%x%x+)}}',
7487     function (n)
7488         return u.char(tonumber(n, 16))
7489     end)
7490 local froms = {}
7491 for s in string.utfcharacters(from) do
7492     table.insert(froms, s)
7493 end
7494 local cnt = 1
7495 table.insert(Babel.capture_maps, {})
7496 local mlen = table.getn(Babel.capture_maps)
7497 for s in string.utfcharacters(to) do
7498     Babel.capture_maps[mlen][froms[cnt]] = s
7499     cnt = cnt + 1
7500 end
7501 return "]]..Babel.capt_map(m[" .. capno .. "], " ..
7502     (mlen) .. ").." .. "[["
7503 end
7504
7505 -- Create/Extend reversed sorted list of kashida weights:
7506 function Babel.capture_kashida(key, wt)
7507 wt = tonumber(wt)
7508 if Babel.kashida_wts then
7509     for p, q in ipairs(Babel.kashida_wts) do
7510         if wt == q then
7511             break
7512         elseif wt > q then
7513             table.insert(Babel.kashida_wts, p, wt)
7514             break

```

```

7515     elseif table.getn(Babel.kashida_wts) == p then
7516         table.insert(Babel.kashida_wts, wt)
7517     end
7518 end
7519 else
7520     Babel.kashida_wts = { wt }
7521 end
7522 return 'kashida = ' .. wt
7523 end
7524
7525 function Babel.capture_node(id, subtype)
7526     local sbt = 0
7527     for k, v in pairs(node.subtypes(id)) do
7528         if v == subtype then sbt = k end
7529     end
7530     return 'node = {' .. node.id(id) .. ', ' .. sbt .. '}'
7531 end
7532
7533 -- Experimental: applies prehyphenation transforms to a string (letters
7534 -- and spaces).
7535 function Babel.string_prehyphenation(str, locale)
7536     local n, head, last, res
7537     head = node.new(8, 0) -- dummy (hack just to start)
7538     last = head
7539     for s in string.utfvalues(str) do
7540         if s == 20 then
7541             n = node.new(12, 0)
7542         else
7543             n = node.new(29, 0)
7544             n.char = s
7545         end
7546         node.set_attribute(n, Babel.attr_locale, locale)
7547         last.next = n
7548         last = n
7549     end
7550     head = Babel.hyphenate_replace(head, 0)
7551     res = ''
7552     for n in node.traverse(head) do
7553         if n.id == 12 then
7554             res = res .. ' '
7555         elseif n.id == 29 then
7556             res = res .. unicode.utf8.char(n.char)
7557         end
7558     end
7559     tex.print(res)
7560 end
7561 </transforms>

```

10.12 Lua: Auto bidi with basic and basic-r

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

```

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

```

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

Now the basic-r bidi mode. One of the aims is to implement a fast and simple bidi algorithm, with a single loop. I managed to do it for R texts, with a second smaller loop for a special case. The code is still somewhat chaotic, but its behavior is essentially correct. I cannot resist copying the following text from Emacs bidi.c (which also attempts to implement the bidi algorithm with a single loop):

Arrrgh!! The UAX#9 algorithm is too deeply entrenched in the assumption of batch-style processing [...]. May the fleas of a thousand camels infest the armpits of those who design supposedly general-purpose algorithms by looking at their own implementations, and fail to consider other possible implementations!

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, *what* they do and *why*, and not only *how*), but I think (or I hope) I've managed to understand them. In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually *two* R modes (set explicitly in Unicode with RLM and ALM). In babel the dir is set by a higher protocol based on the language/script, which in turn sets the correct dir (<l>, <r> or <al>).

From UAX#9: "Where available, markup should be used instead of the explicit formatting characters". So, this simple version just ignores formatting characters. Actually, most of that annex is devoted to how to handle them.

BD14-BD16 are not implemented. Unicode (and the W3C) are making a great effort to deal with some special problematic cases in "streamed" plain text. I don't think this is the way to go – particular issues should be fixed by a high level interface taking into account the needs of the document. And here is where luatex excels, because everything related to bidi writing is under our control.

```
7562 (*basic-r)
7563 Babel = Babel or {}
7564
7565 Babel.bidi_enabled = true
7566
7567 require('babel-data-bidi.lua')
7568
7569 local characters = Babel.characters
7570 local ranges = Babel.ranges
7571
7572 local DIR = node.id("dir")
7573
7574 local function dir_mark(head, from, to, outer)
7575   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
7576   local d = node.new(DIR)
7577   d.dir = '+' .. dir
7578   node.insert_before(head, from, d)
7579   d = node.new(DIR)
7580   d.dir = '-' .. dir
7581   node.insert_after(head, to, d)
7582 end
7583
7584 function Babel.bidi(head, ispar)
7585   local first_n, last_n      -- first and last char with nums
7586   local last_es             -- an auxiliary 'last' used with nums
7587   local first_d, last_d     -- first and last char in L/R block
7588   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):

```
7589 local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7590 local strong_lr = (strong == 'l') and 'l' or 'r'
7591 local outer = strong
7592
7593 local new_dir = false
7594 local first_dir = false
7595 local inmath = false
7596
7597 local last_lr
```

```

7598
7599 local type_n = ''
7600
7601 for item in node.traverse(head) do
7602
7603   -- three cases: glyph, dir, otherwise
7604   if item.id == node.id'glyph'
7605     or (item.id == 7 and item.subtype == 2) then
7606
7607     local itemchar
7608     if item.id == 7 and item.subtype == 2 then
7609       itemchar = item.replace.char
7610     else
7611       itemchar = item.char
7612     end
7613     local chardata = characters[itemchar]
7614     dir = chardata and chardata.d or nil
7615     if not dir then
7616       for nn, et in ipairs(ranges) do
7617         if itemchar < et[1] then
7618           break
7619         elseif itemchar <= et[2] then
7620           dir = et[3]
7621           break
7622         end
7623       end
7624     end
7625     dir = dir or 'l'
7626     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).

```

7627   if new_dir then
7628     attr_dir = 0
7629     for at in node.traverse(item.attr) do
7630       if at.number == Babel.attr_dir then
7631         attr_dir = at.value & 0x3
7632       end
7633     end
7634     if attr_dir == 1 then
7635       strong = 'r'
7636     elseif attr_dir == 2 then
7637       strong = 'al'
7638     else
7639       strong = 'l'
7640     end
7641     strong_lr = (strong == 'l') and 'l' or 'r'
7642     outer = strong_lr
7643     new_dir = false
7644   end
7645
7646   if dir == 'nsm' then dir = strong end -- W1

```

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

```

7647   dir_real = dir -- We need dir_real to set strong below
7648   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:

```

7649   if strong == 'al' then
7650     if dir == 'en' then dir = 'an' end -- W2

```



```

7651     if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7652     strong_lr = 'r' -- W3
7653     end

```

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

```

7654     elseif item.id == node.id'dir' and not inmath then
7655         new_dir = true
7656         dir = nil
7657     elseif item.id == node.id'math' then
7658         inmath = (item.subtype == 0)
7659     else
7660         dir = nil -- Not a char
7661     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>.

```

7662     if dir == 'en' or dir == 'an' or dir == 'et' then
7663         if dir ~= 'et' then
7664             type_n = dir
7665         end
7666         first_n = first_n or item
7667         last_n = last_es or item
7668         last_es = nil
7669     elseif dir == 'es' and last_n then -- W3+W6
7670         last_es = item
7671     elseif dir == 'cs' then -- it's right - do nothing
7672     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7673         if strong_lr == 'r' and type_n ~= '' then
7674             dir_mark(head, first_n, last_n, 'r')
7675         elseif strong_lr == 'l' and first_d and type_n == 'an' then
7676             dir_mark(head, first_n, last_n, 'r')
7677             dir_mark(head, first_d, last_d, outer)
7678             first_d, last_d = nil, nil
7679         elseif strong_lr == 'l' and type_n ~= '' then
7680             last_d = last_n
7681         end
7682         type_n = ''
7683         first_n, last_n = nil, nil
7684     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:

```

7685     if dir == 'l' or dir == 'r' then
7686         if dir ~= outer then
7687             first_d = first_d or item
7688             last_d = item
7689         elseif first_d and dir ~= strong_lr then
7690             dir_mark(head, first_d, last_d, outer)
7691             first_d, last_d = nil, nil
7692         end
7693     end

```

Mirroring. Each chunk of text in a certain language is considered a “closed” sequence. If <r on r> and <l on l>, it's clearly <r> and <l>, resp'tly, but with other combinations depends on outer. From all these, we select only those resolving <on> → <r>. At the beginning (when last_lr is nil) of an R text, they are mirrored directly. Numbers in R mode are processed. It should not be done, but it doesn't hurt.

```

7694     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7695         item.char = characters[item.char] and
7696             characters[item.char].m or item.char

```

```

7697 elseif (dir or new_dir) and last_lr ~= item then
7698     local mir = outer .. strong_lr .. (dir or outer)
7699     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7700         for ch in node.traverse(node.next(last_lr)) do
7701             if ch == item then break end
7702             if ch.id == node.id'glyph' and characters[ch.char] then
7703                 ch.char = characters[ch.char].m or ch.char
7704             end
7705         end
7706     end
7707 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).

```

7708 if dir == 'l' or dir == 'r' then
7709     last_lr = item
7710     strong = dir_real -- Don't search back - best save now
7711     strong_lr = (strong == 'l') and 'l' or 'r'
7712 elseif new_dir then
7713     last_lr = nil
7714 end
7715 end

```

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

```

7716 if last_lr and outer == 'r' then
7717     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7718         if characters[ch.char] then
7719             ch.char = characters[ch.char].m or ch.char
7720         end
7721     end
7722 end
7723 if first_n then
7724     dir_mark(head, first_n, last_n, outer)
7725 end
7726 if first_d then
7727     dir_mark(head, first_d, last_d, outer)
7728 end

```

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

```

7729 return node.prev(head) or head
7730 end
7731 </basic-r>

```

And here the Lua code for bidi=basic:

```

7732 <*basic>
7733 Babel = Babel or {}
7734
7735 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
7736
7737 Babel.fontmap = Babel.fontmap or {}
7738 Babel.fontmap[0] = {} -- l
7739 Babel.fontmap[1] = {} -- r
7740 Babel.fontmap[2] = {} -- al/an
7741
7742 -- To cancel mirroring. Also OML, OMS, U?
7743 Babel.symbol_fonts = Babel.symbol_fonts or {}
7744 Babel.symbol_fonts[font.id('tenln')] = true
7745 Babel.symbol_fonts[font.id('tenlnw')] = true
7746 Babel.symbol_fonts[font.id('tencirc')] = true
7747 Babel.symbol_fonts[font.id('tencircw')] = true
7748
7749 Babel.bidi_enabled = true
7750 Babel.mirroring_enabled = true

```

```

7751
7752 require('babel-data-bidi.lua')
7753
7754 local characters = Babel.characters
7755 local ranges = Babel.ranges
7756
7757 local DIR = node.id('dir')
7758 local GLYPH = node.id('glyph')
7759
7760 local function insert_implicit(head, state, outer)
7761   local new_state = state
7762   if state.sim and state.eim and state.sim ~= state.eim then
7763     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
7764     local d = node.new(DIR)
7765     d.dir = '+' .. dir
7766     node.insert_before(head, state.sim, d)
7767     local d = node.new(DIR)
7768     d.dir = '-' .. dir
7769     node.insert_after(head, state.eim, d)
7770   end
7771   new_state.sim, new_state.eim = nil, nil
7772   return head, new_state
7773 end
7774
7775 local function insert_numeric(head, state)
7776   local new
7777   local new_state = state
7778   if state.san and state.ean and state.san ~= state.ean then
7779     local d = node.new(DIR)
7780     d.dir = '+TLT'
7781     _, new = node.insert_before(head, state.san, d)
7782     if state.san == state.sim then state.sim = new end
7783     local d = node.new(DIR)
7784     d.dir = '-TLT'
7785     _, new = node.insert_after(head, state.ean, d)
7786     if state.ean == state.eim then state.eim = new end
7787   end
7788   new_state.san, new_state.ean = nil, nil
7789   return head, new_state
7790 end
7791
7792 local function glyph_not_symbol_font(node)
7793   if node.id == GLYPH then
7794     return not Babel.symbol_fonts[node.font]
7795   else
7796     return false
7797   end
7798 end
7799
7800 -- TODO - \hbox with an explicit dir can lead to wrong results
7801 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7802 -- was made to improve the situation, but the problem is the 3-dir
7803 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7804 -- well.
7805
7806 function Babel.bidi(head, ispar, hdir)
7807   local d -- d is used mainly for computations in a loop
7808   local prev_d = ''
7809   local new_d = false
7810
7811   local nodes = {}
7812   local outer_first = nil
7813   local inmath = false

```

```

7814
7815 local glue_d = nil
7816 local glue_i = nil
7817
7818 local has_en = false
7819 local first_et = nil
7820
7821 local has_hyperlink = false
7822
7823 local ATDIR = Babel.attr_dir
7824 local attr_d
7825
7826 local save_outer
7827 local temp = node.get_attribute(head, ATDIR)
7828 if temp then
7829     temp = temp & 0x3
7830     save_outer = (temp == 0 and 'l') or
7831                 (temp == 1 and 'r') or
7832                 (temp == 2 and 'al')
7833 elseif ispar then -- Or error? Shouldn't happen
7834     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7835 else -- Or error? Shouldn't happen
7836     save_outer = ('TRT' == hdir) and 'r' or 'l'
7837 end
7838 -- when the callback is called, we are just _after_ the box,
7839 -- and the textdir is that of the surrounding text
7840 -- if not ispar and hdir ~= tex.textdir then
7841 --     save_outer = ('TRT' == hdir) and 'r' or 'l'
7842 -- end
7843 local outer = save_outer
7844 local last = outer
7845 -- 'al' is only taken into account in the first, current loop
7846 if save_outer == 'al' then save_outer = 'r' end
7847
7848 local fontmap = Babel.fontmap
7849
7850 for item in node.traverse(head) do
7851
7852     -- In what follows, #node is the last (previous) node, because the
7853     -- current one is not added until we start processing the neutrals.
7854
7855     -- three cases: glyph, dir, otherwise
7856     if glyph_not_symbol_font(item)
7857         or (item.id == 7 and item.subtype == 2) then
7858
7859         if node.get_attribute(item, ATDIR) == 128 then goto nextnode end
7860
7861         local d_font = nil
7862         local item_r
7863         if item.id == 7 and item.subtype == 2 then
7864             item_r = item.replace -- automatic discs have just 1 glyph
7865         else
7866             item_r = item
7867         end
7868
7869         local chardata = characters[item_r.char]
7870         d = chardata and chardata.d or nil
7871         if not d or d == 'nsm' then
7872             for nn, et in ipairs(ranges) do
7873                 if item_r.char < et[1] then
7874                     break
7875                 elseif item_r.char <= et[2] then
7876                     if not d then d = et[3]

```

```

7877         elseif d == 'nsm' then d_font = et[3]
7878         end
7879         break
7880     end
7881 end
7882 end
7883 d = d or 'l'
7884
7885 -- A short 'pause' in bidi for mapfont
7886 d_font = d_font or d
7887 d_font = (d_font == 'l' and 0) or
7888         (d_font == 'nsm' and 0) or
7889         (d_font == 'r' and 1) or
7890         (d_font == 'al' and 2) or
7891         (d_font == 'an' and 2) or nil
7892 if d_font and fontmap and fontmap[d_font][item_r.font] then
7893     item_r.font = fontmap[d_font][item_r.font]
7894 end
7895
7896 if new_d then
7897     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7898     if inmath then
7899         attr_d = 0
7900     else
7901         attr_d = node.get_attribute(item, ATDIR)
7902         attr_d = attr_d & 0x3
7903     end
7904     if attr_d == 1 then
7905         outer_first = 'r'
7906         last = 'r'
7907     elseif attr_d == 2 then
7908         outer_first = 'r'
7909         last = 'al'
7910     else
7911         outer_first = 'l'
7912         last = 'l'
7913     end
7914     outer = last
7915     has_en = false
7916     first_et = nil
7917     new_d = false
7918 end
7919
7920 if glue_d then
7921     if (d == 'l' and 'l' or 'r') ~= glue_d then
7922         table.insert(nodes, {glue_i, 'on', nil})
7923     end
7924     glue_d = nil
7925     glue_i = nil
7926 end
7927
7928 elseif item.id == DIR then
7929     d = nil
7930
7931     if head ~= item then new_d = true end
7932
7933 elseif item.id == node.id'glue' and item.subtype == 13 then
7934     glue_d = d
7935     glue_i = item
7936     d = nil
7937
7938 elseif item.id == node.id'math' then
7939     inmath = (item.subtype == 0)

```

```

7940
7941 elseif item.id == 8 and item.subtype == 19 then
7942     has_hyperlink = true
7943
7944 else
7945     d = nil
7946 end
7947
7948 -- AL <= EN/ET/ES      -- W2 + W3 + W6
7949 if last == 'al' and d == 'en' then
7950     d = 'an'           -- W3
7951 elseif last == 'al' and (d == 'et' or d == 'es') then
7952     d = 'on'           -- W6
7953 end
7954
7955 -- EN + CS/ES + EN      -- W4
7956 if d == 'en' and #nodes >= 2 then
7957     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7958         and nodes[#nodes-1][2] == 'en' then
7959         nodes[#nodes][2] = 'en'
7960     end
7961 end
7962
7963 -- AN + CS + AN         -- W4 too, because uax9 mixes both cases
7964 if d == 'an' and #nodes >= 2 then
7965     if (nodes[#nodes][2] == 'cs')
7966         and nodes[#nodes-1][2] == 'an' then
7967         nodes[#nodes][2] = 'an'
7968     end
7969 end
7970
7971 -- ET/EN                -- W5 + W7->l / W6->on
7972 if d == 'et' then
7973     first_et = first_et or (#nodes + 1)
7974 elseif d == 'en' then
7975     has_en = true
7976     first_et = first_et or (#nodes + 1)
7977 elseif first_et then    -- d may be nil here !
7978     if has_en then
7979         if last == 'l' then
7980             temp = 'l'    -- W7
7981         else
7982             temp = 'en'  -- W5
7983         end
7984     else
7985         temp = 'on'     -- W6
7986     end
7987     for e = first_et, #nodes do
7988         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
7989     end
7990     first_et = nil
7991     has_en = false
7992 end
7993
7994 -- Force mathdir in math if ON (currently works as expected only
7995 -- with 'l')
7996
7997 if inmath and d == 'on' then
7998     d = ('TRT' == tex.mathdir) and 'r' or 'l'
7999 end
8000
8001 if d then
8002     if d == 'al' then

```

```

8003         d = 'r'
8004         last = 'al'
8005         elseif d == 'l' or d == 'r' then
8006             last = d
8007         end
8008         prev_d = d
8009         table.insert(nodes, {item, d, outer_first})
8010     end
8011
8012     node.set_attribute(item, ATDIR, 128)
8013     outer_first = nil
8014
8015     ::nextnode::
8016
8017 end -- for each node
8018
8019 -- TODO -- repeated here in case EN/ET is the last node. Find a
8020 -- better way of doing things:
8021 if first_et then -- dir may be nil here !
8022     if has_en then
8023         if last == 'l' then
8024             temp = 'l' -- W7
8025         else
8026             temp = 'en' -- W5
8027         end
8028     else
8029         temp = 'on' -- W6
8030     end
8031     for e = first_et, #nodes do
8032         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8033     end
8034 end
8035
8036 -- dummy node, to close things
8037 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8038
8039 ----- NEUTRAL -----
8040
8041 outer = save_outer
8042 last = outer
8043
8044 local first_on = nil
8045
8046 for q = 1, #nodes do
8047     local item
8048
8049     local outer_first = nodes[q][3]
8050     outer = outer_first or outer
8051     last = outer_first or last
8052
8053     local d = nodes[q][2]
8054     if d == 'an' or d == 'en' then d = 'r' end
8055     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
8056
8057     if d == 'on' then
8058         first_on = first_on or q
8059     elseif first_on then
8060         if last == d then
8061             temp = d
8062         else
8063             temp = outer
8064         end
8065         for r = first_on, q - 1 do

```

```

8066     nodes[r][2] = temp
8067     item = nodes[r][1]    -- MIRRORING
8068     if Babel.mirroring_enabled and glyph_not_symbol_font(item)
8069         and temp == 'r' and characters[item.char] then
8070         local font_mode = ''
8071         if item.font > 0 and font.fonts[item.font].properties then
8072             font_mode = font.fonts[item.font].properties.mode
8073         end
8074         if font_mode ~= 'harf' and font_mode ~= 'plug' then
8075             item.char = characters[item.char].m or item.char
8076         end
8077     end
8078 end
8079 first_on = nil
8080 end
8081
8082 if d == 'r' or d == 'l' then last = d end
8083 end
8084
8085 ----- IMPLICIT, REORDER -----
8086
8087 outer = save_outer
8088 last = outer
8089
8090 local state = {}
8091 state.has_r = false
8092
8093 for q = 1, #nodes do
8094
8095     local item = nodes[q][1]
8096
8097     outer = nodes[q][3] or outer
8098
8099     local d = nodes[q][2]
8100
8101     if d == 'nsm' then d = last end          -- W1
8102     if d == 'en' then d = 'an' end
8103     local isdir = (d == 'r' or d == 'l')
8104
8105     if outer == 'l' and d == 'an' then
8106         state.san = state.san or item
8107         state.ean = item
8108     elseif state.san then
8109         head, state = insert_numeric(head, state)
8110     end
8111
8112     if outer == 'l' then
8113         if d == 'an' or d == 'r' then      -- im -> implicit
8114             if d == 'r' then state.has_r = true end
8115             state.sim = state.sim or item
8116             state.eim = item
8117         elseif d == 'l' and state.sim and state.has_r then
8118             head, state = insert_implicit(head, state, outer)
8119         elseif d == 'l' then
8120             state.sim, state.eim, state.has_r = nil, nil, false
8121         end
8122     else
8123         if d == 'an' or d == 'l' then
8124             if nodes[q][3] then -- nil except after an explicit dir
8125                 state.sim = item -- so we move sim 'inside' the group
8126             else
8127                 state.sim = state.sim or item
8128             end
8129         end
8130     end
8131 end

```



```

8129     state.eim = item
8130     elseif d == 'r' and state.sim then
8131         head, state = insert_implicit(head, state, outer)
8132     elseif d == 'r' then
8133         state.sim, state.eim = nil, nil
8134     end
8135 end
8136
8137 if isdir then
8138     last = d          -- Don't search back - best save now
8139 elseif d == 'on' and state.san then
8140     state.san = state.san or item
8141     state.ean = item
8142 end
8143
8144 end
8145
8146 head = node.prev(head) or head
8147
8148 ----- FIX HYPERLINKS -----
8149
8150 if has_hyperlink then
8151     local flag, linking = 0, 0
8152     for item in node.traverse(head) do
8153         if item.id == DIR then
8154             if item.dir == '+TRT' or item.dir == '+TLT' then
8155                 flag = flag + 1
8156             elseif item.dir == '-TRT' or item.dir == '-TLT' then
8157                 flag = flag - 1
8158             end
8159         elseif item.id == 8 and item.subtype == 19 then
8160             linking = flag
8161         elseif item.id == 8 and item.subtype == 20 then
8162             if linking > 0 then
8163                 if item.prev.id == DIR and
8164                     (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
8165                     d = node.new(DIR)
8166                     d.dir = item.prev.dir
8167                     node.remove(head, item.prev)
8168                     node.insert_after(head, item, d)
8169                 end
8170             end
8171             linking = 0
8172         end
8173     end
8174 end
8175
8176 return head
8177 end
8178 -- Make sure anything is marked as 'bidi done' (including nodes inserted
8179 -- after the babel algorithm).
8180 function Babel.unset_atdir(head)
8181     local ATDIR = Babel.attr_dir
8182     for item in node.traverse(head) do
8183         node.set_attribute(item, ATDIR, 128)
8184     end
8185     return head
8186 end
8187 \(/basic\)

```

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.

```
8188 (*nil)
8189 \ProvidesLanguage{nil}[<@date@> v<@version@> Nil language]
8190 \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.

```
8191 \ifx\l@nil\undefined
8192   \newlanguage\l@nil
8193   \@namedef{bbl@hyphendata@the\l@nil}{\{}}% Remove warning
8194   \let\bbl@elt\relax
8195   \edef\bbl@languages{% Add it to the list of languages
8196     \bbl@languages\bbl@elt{nil}{the\l@nil}\{}}
8197 \fi
```

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

```
8198 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
\datenil 8199 \let\captionnil\@empty
8200 \let\datenil\@empty
```

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

```
8201 \def\bbl@inidata@nil{%
8202   \bbl@elt{identification}{tag.ini}{und}%
8203   \bbl@elt{identification}{load.level}{0}%
8204   \bbl@elt{identification}{charset}{utf8}%
8205   \bbl@elt{identification}{version}{1.0}%
8206   \bbl@elt{identification}{date}{2022-05-16}%
8207   \bbl@elt{identification}{name.local}{nil}%
8208   \bbl@elt{identification}{name.english}{nil}%
8209   \bbl@elt{identification}{name.babel}{nil}%
8210   \bbl@elt{identification}{tag.bcp47}{und}%
8211   \bbl@elt{identification}{language.tag.bcp47}{und}%
8212   \bbl@elt{identification}{tag.opentype}{dflt}%
8213   \bbl@elt{identification}{script.name}{Latin}%
8214   \bbl@elt{identification}{script.tag.bcp47}{Latn}%
8215   \bbl@elt{identification}{script.tag.opentype}{DFLT}%
8216   \bbl@elt{identification}{level}{1}%
8217   \bbl@elt{identification}{encodings}{}}%
```

```

8218 \bbl@elt{identification}{derivate}{no}}
8219 \@namedef{bbl@tbc@nil}{und}
8220 \@namedef{bbl@lbc@nil}{und}
8221 \@namedef{bbl@cas@nil}{und} % TODO
8222 \@namedef{bbl@lotf@nil}{dflt}
8223 \@namedef{bbl@elname@nil}{nil}
8224 \@namedef{bbl@lname@nil}{nil}
8225 \@namedef{bbl@sname@nil}{Latin}
8226 \@namedef{bbl@sname@nil}{Latin}
8227 \@namedef{bbl@sbc@nil}{Latn}
8228 \@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.

```

8229 \ldf@finish{nil}
8230 </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.

```

8231 <<Compute Julian day>> ≡
8232 \def\bbl@fpmo#1#2{(#1-#2*floor(#1/#2))}
8233 \def\bbl@cs@gregleap#1{%
8234   (\bbl@fpmo{#1}{4} == 0) &&
8235   (!( (\bbl@fpmo{#1}{100} == 0) && (\bbl@fpmo{#1}{400} != 0) ))}
8236 \def\bbl@cs@jd#1#2#3{% year, month, day
8237   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
8238     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
8239     floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) +
8240     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3} }}
8241 <</Compute Julian day>>

```

13.1 Islamic

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

```

8242 <ca-islamic>
8243 \ExplSyntaxOn
8244 <@Compute Julian day@>
8245 % == islamic (default)
8246 % Not yet implemented
8247 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{}

```

The Civil calendar:

```

8248 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
8249   ((#3 + ceil(29.5 * (#2 - 1)) +
8250   (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
8251   1948439.5) - 1) }
8252 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
8253 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
8254 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
8255 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
8256 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
8257 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
8258   \edef\bbl@tempa{%
8259     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
8260   \edef#5{%
8261     \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
8262   \edef#6{\fp_eval:n{
8263     min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%

```

```
8264 \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).

```
8265 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
8266 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
8267 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
8268 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
8269 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
8270 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
8271 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
8272 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
8273 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
8274 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
8275 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
8276 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
8277 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
8278 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8279 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8280 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8281 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
8282 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
8283 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8284 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8285 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8286 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8287 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8288 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
8289 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8290 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8291 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
8292 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8293 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8294 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8295 65401,65431,65460,65490,65520}
8296 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
8297 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
8298 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
8299 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
8300 \ifnum#2>2014 \ifnum#2<2038
8301 \bbl@afterfi\expandafter@gobble
8302 \fi\fi
8303 {\bbl@error{year-out-range}{2014-2038}}}%
8304 \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
8305 \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8306 \count@\@ne
8307 \bbl@foreach\bbl@cs@umalqura@data{%
8308 \advance\count@\@ne
8309 \ifnum##1>\bbl@tempd\else
8310 \edef\bbl@tempe{\the\count@}%
8311 \edef\bbl@tempb{##1}%
8312 \fi}%
8313 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
8314 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1) / 12) }}% annus
8315 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
8316 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
8317 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}}
8318 \ExplSyntaxOff
8319 \bbl@add\bbl@precalendar{%
8320 \bbl@replace\bbl@ld@calendar{-civil}{}%
8321 \bbl@replace\bbl@ld@calendar{-umalqura}{}}
```

```

8322 \bbl@replace\bbl@ld@calendar{+}{}%
8323 \bbl@replace\bbl@ld@calendar{-}{%}
8324 </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

```

8325 <*ca-hebrew>
8326 \newcount\bbl@cntcommon
8327 \def\bbl@remainder#1#2#3{%
8328   #3=#1\relax
8329   \divide #3 by #2\relax
8330   \multiply #3 by -#2\relax
8331   \advance #3 by #1\relax}%
8332 \newif\ifbbl@divisible
8333 \def\bbl@checkifdivisible#1#2{%
8334   {\countdef\tmp=0
8335    \bbl@remainder{#1}{#2}{\tmp}%
8336    \ifnum \tmp=0
8337      \global\bbl@divisibletrue
8338    \else
8339      \global\bbl@divisiblefalse
8340    \fi}}
8341 \newif\ifbbl@gregleap
8342 \def\bbl@ifgregleap#1{%
8343   \bbl@checkifdivisible{#1}{4}%
8344   \ifbbl@divisible
8345     \bbl@checkifdivisible{#1}{100}%
8346     \ifbbl@divisible
8347       \bbl@checkifdivisible{#1}{400}%
8348       \ifbbl@divisible
8349         \bbl@gregleaptrue
8350       \else
8351         \bbl@gregleapfalse
8352       \fi
8353     \else
8354       \bbl@gregleaptrue
8355     \fi
8356   \else
8357     \bbl@gregleapfalse
8358   \fi
8359   \ifbbl@gregleap}
8360 \def\bbl@gregdayspriormonths#1#2#3{%
8361   {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
8362     181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
8363   \bbl@ifgregleap{#2}%
8364   \ifnum #1 > 2
8365     \advance #3 by 1
8366   \fi
8367   \fi
8368   \global\bbl@cntcommon=#3}%
8369   #3=\bbl@cntcommon}
8370 \def\bbl@gregdaysprioryears#1#2{%
8371   {\countdef\tmpc=4
8372    \countdef\tmpb=2
8373    \tmpb=#1\relax
8374    \advance \tmpb by -1
8375    \tmpc=\tmpb
8376    \multiply \tmpc by 365
8377    #2=\tmpc
8378    \tmpc=\tmpb

```

```

8379 \divide \tmpc by 4
8380 \advance #2 by \tmpc
8381 \tmpc=\tmpb
8382 \divide \tmpc by 100
8383 \advance #2 by -\tmpc
8384 \tmpc=\tmpb
8385 \divide \tmpc by 400
8386 \advance #2 by \tmpc
8387 \global\bbbl@cntcommon=#2\relax}%
8388 #2=\bbbl@cntcommon}
8389 \def\bbbl@absfromgreg#1#2#3#4{%
8390 {\countdef\tmpd=0
8391 #4=#1\relax
8392 \bbbl@gregdayspriormonths{#2}{#3}{\tmpd}%
8393 \advance #4 by \tmpd
8394 \bbbl@gregdaysprioryears{#3}{\tmpd}%
8395 \advance #4 by \tmpd
8396 \global\bbbl@cntcommon=#4\relax}%
8397 #4=\bbbl@cntcommon}
8398 \newif\ifbbbl@hebrleap
8399 \def\bbbl@checkleaphebryear#1{%
8400 {\countdef\tmpa=0
8401 \countdef\tmpb=1
8402 \tmpa=#1\relax
8403 \multiply \tmpa by 7
8404 \advance \tmpa by 1
8405 \bbbl@remainder{\tmpa}{19}{\tmpb}%
8406 \ifnum \tmpb < 7
8407 \global\bbbl@hebrleaptrue
8408 \else
8409 \global\bbbl@hebrleapfalse
8410 \fi}}
8411 \def\bbbl@hebrelapsedmonths#1#2{%
8412 {\countdef\tmpa=0
8413 \countdef\tmpb=1
8414 \countdef\tmpc=2
8415 \tmpa=#1\relax
8416 \advance \tmpa by -1
8417 #2=\tmpa
8418 \divide #2 by 19
8419 \multiply #2 by 235
8420 \bbbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8421 \tmpc=\tmpb
8422 \multiply \tmpb by 12
8423 \advance #2 by \tmpb
8424 \multiply \tmpc by 7
8425 \advance \tmpc by 1
8426 \divide \tmpc by 19
8427 \advance #2 by \tmpc
8428 \global\bbbl@cntcommon=#2}%
8429 #2=\bbbl@cntcommon}
8430 \def\bbbl@hebrelapseday#1#2{%
8431 {\countdef\tmpa=0
8432 \countdef\tmpb=1
8433 \countdef\tmpc=2
8434 \bbbl@hebrelapsedmonths{#1}{#2}%
8435 \tmpa=#2\relax
8436 \multiply \tmpa by 13753
8437 \advance \tmpa by 5604
8438 \bbbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8439 \divide \tmpa by 25920
8440 \multiply #2 by 29
8441 \advance #2 by 1

```

```

8442 \advance #2 by \tmpa
8443 \bbl@remainder{#2}{7}{\tmpa}%
8444 \ifnum \tmpc < 19440
8445     \ifnum \tmpc < 9924
8446     \else
8447         \ifnum \tmpa=2
8448             \bbl@checkleaphebryear{#1}% of a common year
8449             \ifbbl@hebrleap
8450             \else
8451                 \advance #2 by 1
8452             \fi
8453         \fi
8454     \fi
8455     \ifnum \tmpc < 16789
8456     \else
8457         \ifnum \tmpa=1
8458             \advance #1 by -1
8459             \bbl@checkleaphebryear{#1}% at the end of leap year
8460             \ifbbl@hebrleap
8461                 \advance #2 by 1
8462             \fi
8463         \fi
8464     \fi
8465 \else
8466     \advance #2 by 1
8467 \fi
8468 \bbl@remainder{#2}{7}{\tmpa}%
8469 \ifnum \tmpa=0
8470     \advance #2 by 1
8471 \else
8472     \ifnum \tmpa=3
8473         \advance #2 by 1
8474     \else
8475         \ifnum \tmpa=5
8476             \advance #2 by 1
8477         \fi
8478     \fi
8479 \fi
8480 \global\bbl@cntcommon=#2\relax}%
8481 #2=\bbl@cntcommon}
8482 \def\bbl@daysinhebryear#1#2{%
8483     {\countdef\tmpe=12
8484     \bbl@hebrlapseddays{#1}{\tmpe}%
8485     \advance #1 by 1
8486     \bbl@hebrlapseddays{#1}{#2}%
8487     \advance #2 by -\tmpe
8488     \global\bbl@cntcommon=#2}%
8489 #2=\bbl@cntcommon}
8490 \def\bbl@hebrdayspriormonths#1#2#3{%
8491     {\countdef\tmpf= 14
8492     #3=\ifcase #1\relax
8493         0 \or
8494         0 \or
8495         30 \or
8496         59 \or
8497         89 \or
8498         118 \or
8499         148 \or
8500         148 \or
8501         177 \or
8502         207 \or
8503         236 \or
8504         266 \or

```

```

8505         295 \or
8506         325 \or
8507         400
8508     \fi
8509     \bbl@checkleaphebrewyear{#2}%
8510     \ifbbl@hebrleap
8511         \ifnum #1 > 6
8512             \advance #3 by 30
8513         \fi
8514     \fi
8515     \bbl@daysinhebrewyear{#2}{\tmpf}%
8516     \ifnum #1 > 3
8517         \ifnum \tmpf=353
8518             \advance #3 by -1
8519         \fi
8520         \ifnum \tmpf=383
8521             \advance #3 by -1
8522         \fi
8523     \fi
8524     \ifnum #1 > 2
8525         \ifnum \tmpf=355
8526             \advance #3 by 1
8527         \fi
8528         \ifnum \tmpf=385
8529             \advance #3 by 1
8530         \fi
8531     \fi
8532     \global\bbl@cntcommon=#3\relax}%
8533 #3=\bbl@cntcommon}
8534 \def\bbl@absfromhebr#1#2#3#4{%
8535     {#4=#1\relax
8536     \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8537     \advance #4 by #1\relax
8538     \bbl@hebrrelapseddays{#3}{#1}%
8539     \advance #4 by #1\relax
8540     \advance #4 by -1373429
8541     \global\bbl@cntcommon=#4\relax}%
8542 #4=\bbl@cntcommon}
8543 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8544     {\countdef\tmpx= 17
8545     \countdef\tmpy= 18
8546     \countdef\tmpz= 19
8547     #6=#3\relax
8548     \global\advance #6 by 3761
8549     \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8550     \tmpz=1 \tmpy=1
8551     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8552     \ifnum \tmpx > #4\relax
8553         \global\advance #6 by -1
8554         \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8555     \fi
8556     \advance #4 by -\tmpx
8557     \advance #4 by 1
8558     #5=#4\relax
8559     \divide #5 by 30
8560     \loop
8561         \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8562         \ifnum \tmpx < #4\relax
8563             \advance #5 by 1
8564             \tmpy=\tmpx
8565         \repeat
8566     \global\advance #5 by -1
8567     \global\advance #4 by -\tmpy}}

```



```

8568 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebyear
8569 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8570 \def\bbl@ca@hebrew#1-#2-#3\@#4#5#6{%
8571 \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8572 \bbl@hebrfromgreg
8573 {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8574 {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebyear}%
8575 \edef#4{\the\bbl@hebyear}%
8576 \edef#5{\the\bbl@hebrmonth}%
8577 \edef#6{\the\bbl@hebrday}}
8578 </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).

```

8579 <*ca-persian>
8580 \ExplSyntaxOn
8581 <@Compute Julian day@>
8582 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8583 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8584 \def\bbl@ca@persian#1-#2-#3\@#4#5#6{%
8585 \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8586 \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8587 \bbl@afterfi\expandafter@gobble
8588 \fi\fi
8589 {\bbl@error{year-out-range}{2013-2050}{}}%
8590 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8591 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8592 \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8593 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8594 \ifnum\bbl@tempc<\bbl@tempb
8595 \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8596 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8597 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8598 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8599 \fi
8600 \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8601 \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8602 \edef#5{\fp_eval:n{% set Jalali month
8603 (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8604 \edef#6{\fp_eval:n{% set Jalali day
8605 (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6))}}
8606 \ExplSyntaxOff
8607 </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.

```

8608 <*ca-coptic>
8609 \ExplSyntaxOn
8610 <@Compute Julian day@>
8611 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
8612 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8613 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8614 \edef#4{\fp_eval:n{%
8615 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8616 \edef\bbl@tempc{\fp_eval:n{%

```

```

8617 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8618 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8619 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}
8620 \ExplSyntaxOff
8621 </ca-coptic>
8622 <*ca-ethiopic>
8623 \ExplSyntaxOn
8624 <@Compute Julian day@>
8625 \def\bbl@ca@ethiopic#1-#2-#3\@@#4#5#6{%
8626 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8627 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8628 \edef#4{\fp_eval:n{%
8629 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8630 \edef\bbl@tempc{\fp_eval:n{%
8631 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8632 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8633 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}
8634 \ExplSyntaxOff
8635 </ca-ethiopic>

```

13.5 Buddhist

That's very simple.

```

8636 <*ca-buddhist>
8637 \def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
8638 \edef#4{\number\numexpr#1+543\relax}%
8639 \edef#5{#2}%
8640 \edef#6{#3}}
8641 </ca-buddhist>
8642 %
8643 \subsection{Chinese}
8644 %
8645 % Brute force, with the Julian day of first day of each month. The
8646 % table has been computed with the help of \textsf{python-lunardate} by
8647 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
8648 % is 2015-2044.
8649 %
8650 % \begin{macrocode}
8651 <*ca-chinese>
8652 \ExplSyntaxOn
8653 <@Compute Julian day@>
8654 \def\bbl@ca@chinese#1-#2-#3\@@#4#5#6{%
8655 \edef\bbl@tempd{\fp_eval:n{%
8656 \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
8657 \count@z@
8658 \@tempcnta=2015
8659 \bbl@foreach\bbl@cs@chinese@data{%
8660 \ifnum##1>\bbl@tempd\else
8661 \advance\count@\@ne
8662 \ifnum\count@>12
8663 \count@\@ne
8664 \advance\@tempcnta\@ne\fi
8665 \bbl@xin@{,##1,}{,\bbl@cs@chinese@leap,}%
8666 \ifin@
8667 \advance\count@\m@ne
8668 \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
8669 \else
8670 \edef\bbl@tempe{\the\count@}%
8671 \fi
8672 \edef\bbl@tempb{##1}%
8673 \fi}%
8674 \edef#4{\the\@tempcnta}%
8675 \edef#5{\bbl@tempe}%

```

```

8676 \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
8677 \def\bbl@cs@chinese@leap{%
8678 885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
8679 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
8680 354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
8681 768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
8682 1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
8683 1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
8684 1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
8685 2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
8686 2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
8687 2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
8688 3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
8689 3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
8690 3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
8691 4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
8692 4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
8693 5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
8694 5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
8695 5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
8696 6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
8697 6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
8698 6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
8699 7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
8700 7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
8701 7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
8702 8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8703 8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8704 8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
8705 9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
8706 9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
8707 10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
8708 10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
8709 10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
8710 10896,10926,10956,10986,11015,11045,11074,11103}
8711 \ExplSyntaxOff
8712 </ca-chinese>

```

14 Support for Plain T_EX (plain.def)

14.1 Not renaming hyphen.tex

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based T_EX-format. When asked he responded:

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

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

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

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

```

8713 (*bplain | blplain)
8714 \catcode`\{=1 % left brace is begin-group character
8715 \catcode`\}=2 % right brace is end-group character
8716 \catcode`\#=6 % hash mark is macro parameter character

```

If a file called `hyphen.cfg` can be found, we make sure that *it* will be read instead of the file `hyphen.tex`. We do this by first saving the original meaning of `\input` (and I use a one letter control sequence for that so as not to waste multi-letter control sequence on this in the format).

```
8717 \openin 0 hyphen.cfg
8718 \ifeof0
8719 \else
8720 \let\input
```

Then `\input` is defined to forget about its argument and load `hyphen.cfg` instead. Once that's done the original meaning of `\input` can be restored and the definition of `\a` can be forgotten.

```
8721 \def\input #1 {%
8722 \let\input\input
8723 \a hyphen.cfg
8724 \let\input\input
8725 }
8726 \fi
8727 </bplain | bplain>
```

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

```
8728 <bplain>\a plain.tex
8729 <bplain>\a lplain.tex
```

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

```
8730 <bplain>\def\fmtname{babel-plain}
8731 <bplain>\def\fmtname{babel-lplain}
```

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

14.2 Emulating some \LaTeX features

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

```
8732 <<*Emulate LaTeX>> ≡
8733 \def\@empty{}
8734 \def\loadlocalcfg#1{%
8735 \openin0#1.cfg
8736 \ifeof0
8737 \closein0
8738 \else
8739 \closein0
8740 {\immediate\writel6{*****}%
8741 \immediate\writel6{* Local config file #1.cfg used}%
8742 \immediate\writel6{*}%
8743 }
8744 \input #1.cfg\relax
8745 \fi
8746 \@endofldf}
```

14.3 General tools

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

```
8747 \long\def\@firstofone#1{#1}
8748 \long\def\@firstoftwo#1#2{#1}
8749 \long\def\@secondoftwo#1#2{#2}
8750 \def\@nnil{\@nil}
8751 \def\@gobbletwo#1#2{}
8752 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
```

```

8753 \def\@star@or@long#1{%
8754   \ifstar
8755   {\let\l@ngrel@x\relax#1}%
8756   {\let\l@ngrel@x\long#1}}
8757 \let\l@ngrel@x\relax
8758 \def\@car#1#2\@nil{#1}
8759 \def\@cdr#1#2\@nil{#2}
8760 \let\@typeset@protect\relax
8761 \let\protected@edef\edef
8762 \long\def\@gobble#1{}
8763 \edef\@backslashchar{\expandafter\@gobble\string\}
8764 \def\strip@prefix#1>{}
8765 \def\g@addto@macro#1#2{%
8766   \toks@\expandafter{#1#2}%
8767   \xdef#1{\the\toks@}}
8768 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8769 \def\@nameuse#1{\csname #1\endcsname}
8770 \def\@ifundefined#1{%
8771   \expandafter\ifx\csname#1\endcsname\relax
8772     \expandafter\@firstoftwo
8773   \else
8774     \expandafter\@secondoftwo
8775   \fi}
8776 \def\@expandtwoargs#1#2#3{%
8777   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8778 \def\zap@space#1 #2{%
8779   #1%
8780   \ifx#2\@empty\else\expandafter\zap@space\fi
8781   #2}
8782 \let\bbl@trace\@gobble
8783 \def\bbl@error#1{% Implicit #2#3#4}
8784 \begingroup
8785   \catcode`\=0 \catcode`\==12 \catcode`\`=12
8786   \catcode`\^^M=5 \catcode`\%=14
8787   \input errbabel.def
8788 \endgroup
8789 \bbl@error{#1}}
8790 \def\bbl@warning#1{%
8791   \begingroup
8792     \newlinechar=\^^J
8793     \def\{\^^J(babel) }%
8794     \message{\#1}%
8795   \endgroup}
8796 \let\bbl@infowarn\bbl@warning
8797 \def\bbl@info#1{%
8798   \begingroup
8799     \newlinechar=\^^J
8800     \def\{\^^J}%
8801     \wlog{#1}%
8802   \endgroup}

```

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

```

8803 \ifx\@preamblecmds\undefined
8804   \def\@preamblecmds{}
8805 \fi
8806 \def\@onlypreamble#1{%
8807   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8808     \@preamblecmds\do#1}}
8809 \@onlypreamble\@onlypreamble

```

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

```

8810 \def\begindocument{%
8811   \@begindocumenthook

```

```

8812 \global\let\@begindocumenthook\@undefined
8813 \def\do##1{\global\let##1\@undefined}%
8814 \@preamblecmds
8815 \global\let\do\noexpand}

8816 \ifx\@begindocumenthook\@undefined
8817 \def\@begindocumenthook{}
8818 \fi
8819 \@onlypreamble\@begindocumenthook
8820 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

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

```

8821 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
8822 \@onlypreamble\AtEndOfPackage
8823 \def\@endofldf{}
8824 \@onlypreamble\@endofldf
8825 \let\bb1@afterlang\@empty
8826 \chardef\bb1@opt@hyphenmap\z@

```

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

```

8827 \catcode`\&=\z@
8828 \ifx&if@filesw\@undefined
8829 \expandafter\let\csname if@filesw\expandafter\endcsname
8830 \csname iffalse\endcsname
8831 \fi
8832 \catcode`\&=4

```

Mimic \TeX 's commands to define control sequences.

```

8833 \def\newcommand{\@star@or@long\new@command}
8834 \def\new@command#1{%
8835 \@testopt{\@newcommand#1}0}
8836 \def\@newcommand#1[#2]{%
8837 \@ifnextchar [{\@xargdef#1[#2]}%
8838 {\@argdef#1[#2]}}
8839 \long\def\@argdef#1[#2]#3{%
8840 \@yargdef#1\@ne{#2}#3}
8841 \long\def\@xargdef#1[#2][#3]#4{%
8842 \expandafter\def\expandafter#1\expandafter{%
8843 \expandafter\@protected@testopt\expandafter #1%
8844 \csname\string#1\expandafter\endcsname{#3}}%
8845 \expandafter\@yargdef \csname\string#1\endcsname
8846 \tw@{#2}{#4}}
8847 \long\def\@yargdef#1#2#3{%
8848 \@tempcnta#3\relax
8849 \advance \@tempcnta \@ne
8850 \let\@hash@\relax
8851 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
8852 \@tempcntb #2%
8853 \@whilenum\@tempcntb <\@tempcnta
8854 \do{%
8855 \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
8856 \advance\@tempcntb \@ne}%
8857 \let\@hash@##%
8858 \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
8859 \def\providecommand{\@star@or@long\provide@command}
8860 \def\provide@command#1{%
8861 \begingroup
8862 \escapechar\m@ne\edef\@gtempa{\string#1}%
8863 \endgroup
8864 \expandafter\ifundefined\@gtempa
8865 {\def\reserved@a{\new@command#1}}%

```

```

8866   {\let\reserved@a\relax
8867     \def\reserved@a{\newcommand\reserved@a}}%
8868   \reserved@a}%

8869 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
8870 \def\declare@robustcommand#1{%
8871   \edef\reserved@a{\string#1}%
8872   \def\reserved@b{#1}%
8873   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
8874   \edef#1{%
8875     \ifx\reserved@a\reserved@b
8876       \noexpand\x@protect
8877       \noexpand#1%
8878     \fi
8879     \noexpand\protect
8880     \expandafter\noexpand\csname
8881       \expandafter\@gobble\string#1 \endcsname
8882   }%
8883   \expandafter\newcommand\csname
8884     \expandafter\@gobble\string#1 \endcsname
8885 }
8886 \def\x@protect#1{%
8887   \ifx\protect\@typeset@protect\else
8888     \@x@protect#1%
8889   \fi
8890 }
8891 \catcode`\&=\z@ % Trick to hide conditionals
8892 \def\@x@protect#1&#2#3{\fi\protect#1}

```

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

```

8893 \def\bbl@tempa{\csname newif\endcsname&in@}
8894 \catcode`\&=4
8895 \ifx\in@\undefined
8896   \def\in@#1#2{%
8897     \def\in@##1#1##2##3\in@{%
8898       \ifx\in@##2\in@false\else\in@true\fi}%
8899     \in@#2#1\in@\in@@}
8900 \else
8901   \let\bbl@tempa\@empty
8902 \fi
8903 \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).

```

8904 \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.

```

8905 \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.

```

8906 \ifx\@tempcnta\undefined
8907   \csname newcount\endcsname\@tempcnta\relax
8908 \fi
8909 \ifx\@tempcntb\undefined
8910   \csname newcount\endcsname\@tempcntb\relax
8911 \fi

```

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

```

8912 \ifx\bye\@undefined
8913   \advance\count10 by -2\relax
8914 \fi
8915 \ifx\@ifnextchar\@undefined
8916   \def\@ifnextchar#1#2#3{%
8917     \let\reserved@d=#1%
8918     \def\reserved@a{#2}\def\reserved@b{#3}%
8919     \futurelet\@let@token\@ifnch}
8920 \def\@ifnch{%
8921   \ifx\@let@token\@sptoken
8922     \let\reserved@c\@xifnch
8923   \else
8924     \ifx\@let@token\reserved@d
8925       \let\reserved@c\reserved@a
8926     \else
8927       \let\reserved@c\reserved@b
8928     \fi
8929   \fi
8930   \reserved@c}
8931 \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
8932 \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
8933 \fi
8934 \def\@testopt#1#2{%
8935   \@ifnextchar[#{1}{#1[#2]}}
8936 \def\@protected@testopt#1{%
8937   \ifx\protect\@typeset@protect
8938     \expandafter\@testopt
8939   \else
8940     \@x@protect#1%
8941   \fi}
8942 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8943   #2\relax}\fi}
8944 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8945   \else\expandafter\@gobble\fi{#1}}

```

14.4 Encoding related macros

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

```

8946 \def\DeclareTextCommand{%
8947   \@dec@text@cmd\providecommand
8948 }
8949 \def\ProvideTextCommand{%
8950   \@dec@text@cmd\providecommand
8951 }
8952 \def\DeclareTextSymbol#1#2#3{%
8953   \@dec@text@cmd\chardef#1{#2}#3\relax
8954 }
8955 \def\@dec@text@cmd#1#2#3{%
8956   \expandafter\def\expandafter#2%
8957     \expandafter{%
8958       \csname#3-cmd\expandafter\endcsname
8959       \expandafter#2%
8960       \csname#3\string#2\endcsname
8961     }%
8962 %   \let\@ifdefinable\@rc@ifdefinable
8963   \expandafter#1\csname#3\string#2\endcsname
8964 }
8965 \def\@current@cmd#1{%
8966   \ifx\protect\@typeset@protect\else
8967     \noexpand#1\expandafter\@gobble

```



```

8968 \fi
8969 }
8970 \def\@changed@cmd#1#2{%
8971 \ifx\protect\@typeset@protect
8972 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8973 \expandafter\ifx\csname ?\string#1\endcsname\relax
8974 \expandafter\def\csname ?\string#1\endcsname{%
8975 \@changed@x@err{#1}%
8976 }%
8977 \fi
8978 \global\expandafter\let
8979 \csname\cf@encoding \string#1\expandafter\endcsname
8980 \csname ?\string#1\endcsname
8981 \fi
8982 \csname\cf@encoding\string#1%
8983 \expandafter\endcsname
8984 \else
8985 \noexpand#1%
8986 \fi
8987 }
8988 \def\@changed@x@err#1{%
8989 \errhelp{Your command will be ignored, type <return> to proceed}%
8990 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8991 \def\DeclareTextCommandDefault#1{%
8992 \DeclareTextCommand#1?%
8993 }
8994 \def\ProvideTextCommandDefault#1{%
8995 \ProvideTextCommand#1?%
8996 }
8997 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8998 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8999 \def\DeclareTextAccent#1#2#3{%
9000 \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
9001 }
9002 \def\DeclareTextCompositeCommand#1#2#3#4{%
9003 \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
9004 \edef\reserved@b{\string##1}%
9005 \edef\reserved@c{%
9006 \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
9007 \ifx\reserved@b\reserved@c
9008 \expandafter\expandafter\expandafter\ifx
9009 \expandafter\@car\reserved@a\relax\relax\@nil
9010 \@text@composite
9011 \else
9012 \edef\reserved@b##1{%
9013 \def\expandafter\noexpand
9014 \csname#2\string#1\endcsname###1{%
9015 \noexpand\@text@composite
9016 \expandafter\noexpand\csname#2\string#1\endcsname
9017 ###1\noexpand\@empty\noexpand\@text@composite
9018 {##1}%
9019 }%
9020 }%
9021 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
9022 \fi
9023 \expandafter\def\csname\expandafter\string\csname
9024 #2\endcsname\string#1-\string#3\endcsname{#4}
9025 \else
9026 \errhelp{Your command will be ignored, type <return> to proceed}%
9027 \errmessage{\string\DeclareTextCompositeCommand\space used on
9028 inappropriate command \protect#1}
9029 \fi
9030 }

```

```

9031 \def\@text@composite#1#2#3\@text@composite{%
9032   \expandafter\@text@composite@x
9033     \csname\string#1-\string#2\endcsname
9034 }
9035 \def\@text@composite@x#1#2{%
9036   \ifx#1\relax
9037     #2%
9038   \else
9039     #1%
9040   \fi
9041 }
9042 %
9043 \def\@strip@args#1:#2-#3\@strip@args{#2}
9044 \def\DeclareTextComposite#1#2#3#4{%
9045   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
9046   \bgroup
9047     \lccode`\@=#4%
9048     \lowercase{%
9049   \egroup
9050     \reserved@a @%
9051   }%
9052 }
9053 %
9054 \def\UseTextSymbol#1#2{#2}
9055 \def\UseTextAccent#1#2#3{}
9056 \def\@use@text@encoding#1{}
9057 \def\DeclareTextSymbolDefault#1#2{%
9058   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
9059 }
9060 \def\DeclareTextAccentDefault#1#2{%
9061   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
9062 }
9063 \def\cf@encoding{OT1}

```

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

```

9064 \DeclareTextAccent{"}{OT1}{127}
9065 \DeclareTextAccent{\'}{OT1}{19}
9066 \DeclareTextAccent{\^}{OT1}{94}
9067 \DeclareTextAccent{\`}{OT1}{18}
9068 \DeclareTextAccent{\~}{OT1}{126}

```

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

```

9069 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
9070 \DeclareTextSymbol{\textquotedblright}{OT1}{`\`"}
9071 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
9072 \DeclareTextSymbol{\textquoteright}{OT1}{``'}
9073 \DeclareTextSymbol{\i}{OT1}{16}
9074 \DeclareTextSymbol{\ss}{OT1}{25}

```

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

```

9075 \ifx\scriptsize\@undefined
9076   \let\scriptsize\sevenrm
9077 \fi

```

And a few more “dummy” definitions.

```

9078 \def\language{english}%
9079 \let\bbl@opt@shorthands\@nnil
9080 \def\bbl@ifshorthand#1#2#3{#2}%
9081 \let\bbl@language@opts\@empty
9082 \let\bbl@ensureinfo\@gobble
9083 \let\bbl@provide@locale\relax
9084 \ifx\babeloptionstrings\@undefined

```

```

9085 \let\bbl@opt@strings@nnil
9086 \else
9087 \let\bbl@opt@strings\babeloptionstrings
9088 \fi
9089 \def\BabelStringsDefault{generic}
9090 \def\bbl@tempa{normal}
9091 \ifx\babeloptionmath\bbl@tempa
9092 \def\bbl@mathnormal{\noexpand\textormath}
9093 \fi
9094 \def\AfterBabelLanguage#1#2{}
9095 \ifx\BabelModifiers@undefined\let\BabelModifiers\relax\fi
9096 \let\bbl@afterlang\relax
9097 \def\bbl@opt@safe{BR}
9098 \ifx\@uclclist@undefined\let\@uclclist@empty\fi
9099 \ifx\bbl@trace@undefined\def\bbl@trace#1{}\fi
9100 \expandafter\newif\csname ifbbl@single\endcsname
9101 \chardef\bbl@bidimode\z@
9102 <</Emulate LaTeX>>

A proxy file:
9103 <*\plain>
9104 \input babel.def
9105 </\plain>

```

15 Acknowledgements

In the initial stages of the development of babel, Bernd Raichle provided many helpful suggestions and Michel Goossens supplied contributions for many languages. Ideas from Nico Poppelier, Piet van Oostrum and many others have been used. Paul Wackers and Werenfried Spit helped find and repair bugs.

More recently, there are significant contributions by Salim Bou, Ulrike Fischer and Udi Fogiel. There are also many contributors for specific languages, which are mentioned in the respective files. Without them, babel just wouldn't exist.

References

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