

# Babel

## Code

Version 24.10.63051  
2024/09/21

Javier Bezos  
Current maintainer

Johannes L. Braams  
Original author

Localization and  
internationalization

Unicode

T<sub>E</sub>X

pdfT<sub>E</sub>X

LuaT<sub>E</sub>X

XeT<sub>E</sub>X

# Contents

<b>1</b>	<b>Identification and loading of required files</b>	<b>3</b>
<b>2</b>	<b>locale directory</b>	<b>3</b>
<b>3</b>	<b>Tools</b>	<b>3</b>
3.1	A few core definitions . . . . .	7
3.2	LaTeX: babel.sty (start) . . . . .	8
3.3	base . . . . .	9
3.4	key=value options and other general option . . . . .	10
3.5	Post-process some options . . . . .	11
3.6	Plain: babel.def (start) . . . . .	12
<b>4</b>	<b>babel.sty and babel.def (common)</b>	<b>13</b>
4.1	Selecting the language . . . . .	15
4.2	Short tags . . . . .	24
4.3	Errors . . . . .	25
4.4	Hooks . . . . .	27
4.5	Setting up language files . . . . .	27
4.6	Shorthands . . . . .	30
4.7	Language attributes . . . . .	39
4.8	Support for saving macro definitions . . . . .	40
4.9	Hyphens . . . . .	42
4.10	Multiencoding strings . . . . .	44
4.11	Tailor captions . . . . .	48
4.12	Making glyphs available . . . . .	49
4.12.1	Quotation marks . . . . .	49
4.12.2	Letters . . . . .	51
4.12.3	Shorthands for quotation marks . . . . .	52
4.12.4	Umlauts and tremas . . . . .	53
4.13	Layout . . . . .	54
4.14	Load engine specific macros . . . . .	55
4.15	Creating and modifying languages . . . . .	55
4.16	Main loop in ‘provide’ . . . . .	64
4.17	Processing keys in ini . . . . .	67
4.18	Handle language system . . . . .	74
4.19	Numerals . . . . .	75
4.20	Casing . . . . .	76
4.21	Getting info . . . . .	77
<b>5</b>	<b>Adjusting the Babel behavior</b>	<b>78</b>
5.1	Cross referencing macros . . . . .	80
5.2	Marks . . . . .	83
5.3	Other packages . . . . .	84
5.3.1	ifthen . . . . .	84
5.3.2	varioref . . . . .	85
5.3.3	hhline . . . . .	85
5.4	Encoding and fonts . . . . .	86
5.5	Basic bidi support . . . . .	88
5.6	Local Language Configuration . . . . .	91
5.7	Language options . . . . .	91
<b>6</b>	<b>The kernel of Babel (babel.def, common)</b>	<b>94</b>
<b>7</b>	<b>Error messages</b>	<b>95</b>
<b>8</b>	<b>Loading hyphenation patterns</b>	<b>98</b>
<b>9</b>	<b>xetex + luatex: common stuff</b>	<b>102</b>

<b>10</b>	<b>Hooks for XeTeX and LuaTeX</b>	<b>106</b>
10.1	XeTeX . . . . .	106
<b>11</b>	<b>Support for interchar</b>	<b>107</b>
11.1	Layout . . . . .	109
11.2	8-bit TeX . . . . .	111
11.3	LuaTeX . . . . .	112
11.4	Southeast Asian scripts . . . . .	118
11.5	CJK line breaking . . . . .	119
11.6	Arabic justification . . . . .	121
11.7	Common stuff . . . . .	126
11.8	Automatic fonts and ids switching . . . . .	126
11.9	Bidi . . . . .	132
11.10	Layout . . . . .	134
11.11	Lua: transforms . . . . .	142
11.12	Lua: Auto bidi with basic and basic-r . . . . .	151
<b>12</b>	<b>Data for CJK</b>	<b>162</b>
<b>13</b>	<b>The ‘nil’ language</b>	<b>163</b>
<b>14</b>	<b>Calendars</b>	<b>164</b>
14.1	Islamic . . . . .	164
14.2	Hebrew . . . . .	166
14.3	Persian . . . . .	170
14.4	Coptic and Ethiopic . . . . .	170
14.5	Buddhist . . . . .	171
<b>15</b>	<b>Support for Plain T<sub>E</sub>X (plain.def)</b>	<b>172</b>
15.1	Not renaming hyphen.tex . . . . .	172
15.2	Emulating some L <sup>A</sup> T <sub>E</sub> X features . . . . .	173
15.3	General tools . . . . .	173
15.4	Encoding related macros . . . . .	177
<b>16</b>	<b>Acknowledgements</b>	<b>180</b>

The babel package is being developed incrementally, which means parts of the code are under development and therefore incomplete. Only documented features are considered complete. In other words, use babel in real documents only as documented (except, of course, if you want to explore and test them).

## 1 Identification and loading of required files

*Code documentation is still under revision.*

The babel package after unpacking consists of the following files:

**babel.sty** is the  $\LaTeX$  package, which set options and load language styles.

**babel.def** is loaded by Plain.

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

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

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

There some additional tex, def and lua files.

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

## 2 locale directory

A required component of babel is a set of ini files with basic definitions for about 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=24.10.63051>>
2 <<date=2024/09/21>>
```

**Do not use the following macros in ldf files. They may change in the future.** This applies mainly to those recently added for replacing, trimming and looping. The older ones, like `\bbl@afterfi`, will not change. We define some basic macros which just make the code cleaner. `\bbl@add` is now used internally instead of `\addto` because of the unpredictable behavior of the latter. Used in babel.def and in babel.sty, which means in  $\LaTeX$  is executed twice, but we need them when defining options and babel.def cannot be load until options have been defined. This does not hurt, but should be fixed somehow.

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

```

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

```

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

```

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

```

### **\bbl@afterelse**

**\bbl@afterfi** Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the `\else` and `\fi` parts of an `\if`-statement<sup>1</sup>. These macros will break if another `\if... \fi` statement appears in one of the arguments and it is not enclosed in braces.

```

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

```

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

```

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

```

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

```

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

```

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

**\bbl@ifunset** To check if a macro is defined, we create a new macro, which does the same as `\ifundefined`. However, in an  $\epsilon$ -tex engine, it is based on `\ifcurname`, which is more efficient, and does not waste memory. Defined inside a group, to avoid `\ifcurname` being implicitly set to `\relax` by the `\curname` test.

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

101 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3

```

```

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

```

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

```

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

```

Two further tools. \bbl@ifsamestring first expand its arguments and then compare their expansion (sanitized, so that the catcodes do not matter). \bbl@engine takes the following values: 0 is pdf<sub>TeX</sub>, 1 is luatex, and 2 is xetex. You may use the latter it in your language style if you want.

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

### 3.1 A few core definitions

#### **`\last@language`**

**`\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  $\text{\TeX}$  < 2. Preserved for compatibility.

```
199 <<*Define core switching macros>> ≡
200 \countdef\last@language=19
201 \def\addlanguage{\csname newlanguage\endcsname}
202 <</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 $\text{\LaTeX}$ : `babel.sty` (start)

Here starts the style file for  $\text{\LaTeX}$ . It also takes care of a number of compatibility issues with other packages.

```
203 <*package>
204 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
205 \ProvidesPackage{babel}[<@date@> v<@version@> The Babel package]
```

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

```
206 \ifpackagewith{babel}{debug}
207   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
208    \let\bbl@debug\@firstofone
209    \ifx\directlua\@undefined\else
210      \directlua{ Babel = Babel or {}
211                Babel.debug = true }%
212      \input{babel-debug.tex}%
213    \fi}
214 {\providecommand\bbl@trace[1]{}%
215  \let\bbl@debug\gobble
216  \ifx\directlua\@undefined\else
217    \directlua{ Babel = Babel or {}
218              Babel.debug = false }%
219  \fi}
```

Macros to deal with errors, warnings, etc. Errors are stored in a separate file.

```
220 \def\bbl@error#1{% Implicit #2#3#4
221   \begingroup
222     \catcode`\=0 \catcode`\==12 \catcode`\`=12
223     \input errbabel.def
224   \endgroup
225   \bbl@error{#1}}
226 \def\bbl@warning#1{%
227   \begingroup
228     \def\{\MessageBreak}%
229     \PackageWarning{babel}{#1}%
230   \endgroup}
231 \def\bbl@infowarn#1{%
232   \begingroup
233     \def\{\MessageBreak}%
234     \PackageNote{babel}{#1}%
235   \endgroup}
236 \def\bbl@info#1{%
237   \begingroup
238     \def\{\MessageBreak}%
239     \PackageInfo{babel}{#1}%
240   \endgroup}
```

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.

```

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

```

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

```

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

```

### 3.3 base

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

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

```

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

```
288 \bbl@trace{key=value and another general options}
289 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
290 \def\bbl@tempb#1.#2{% Remove trailing dot
291   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
292 \def\bbl@tempe#1=#2\@@{%
293   \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}}
294 \def\bbl@tempd#1.#2\@nnil{%%^A TODO. Refactor lists?
295   \ifx\@empty#2%
296     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
297   \else
298     \in@{,provide=}{, #1}%
299     \ifin@
300       \edef\bbl@tempc{%
301         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
302     \else
303       \in@{${modifiers$}{$#1$}%%^A TODO. Allow spaces.
304       \ifin@
305         \bbl@tempe#2\@@
306       \else
307         \in@{=}{#1}%
308         \ifin@
309           \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
310         \else
311           \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
312           \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
313         \fi
314       \fi
315     \fi
316   \fi}
317 \let\bbl@tempc\@empty
318 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
319 \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.

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

```

339 \let\bbl@opt@shorthands\@nnil
340 \let\bbl@opt@config\@nnil
341 \let\bbl@opt@main\@nnil
342 \let\bbl@opt@headfoot\@nnil
343 \let\bbl@opt@layout\@nnil
344 \let\bbl@opt@provide\@nnil

```

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

```

345 \def\bbl@tempa#1=#2\bbl@tempa{%
346   \bbl@csarg\ifx{opt#1}\@nnil
347     \bbl@csarg\edef{opt#1}{#2}%
348   \else
349     \bbl@error{bad-package-option}{#1}{#2}{}%
350   \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.

```

351 \let\bbl@language@opts\@empty
352 \DeclareOption*{%
353   \bbl@xin@{\string=}{\CurrentOption}%
354   \ifin@
355     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
356   \else
357     \bbl@add@list\bbl@language@opts{\CurrentOption}%
358   \fi}

```

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

```

359 \ProcessOptions*

```

### 3.5 Post-process some options

```

360 \ifx\bbl@opt@provide\@nnil
361   \let\bbl@opt@provide\@empty %%% MOVE above
362 \else
363   \chardef\bbl@iniflag\@ne
364   \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
365     \in@{,provide,}{, #1,}%
366     \ifin@
367       \def\bbl@opt@provide{#2}%
368     \fi}
369 \fi
370 %

```

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=...`

```

371 \bbl@trace{Conditional loading of shorthands}
372 \def\bbl@sh@string#1{%
373   \ifx#1\@empty\else
374     \ifx#1\string~%
375     \else\ifx#1c\string,%
376     \else\string#1%
377   \fi\fi
378   \expandafter\bbl@sh@string
379 \fi}
380 \ifx\bbl@opt@shorthands\@nnil
381   \def\bbl@ifshorthand#1#2#3{#2}%
382 \else\ifx\bbl@opt@shorthands\@empty
383   \def\bbl@ifshorthand#1#2#3{#3}%
384 \else

```

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

```

385 \def\bbl@ifshorthand#1{%
386   \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
387   \ifin@
388     \expandafter\@firstoftwo
389   \else
390     \expandafter\@secondoftwo
391   \fi}

```

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

```

392 \edef\bbl@opt@shorthands{%
393   \expandafter\bbl@sh@string\bbl@opt@shorthands\@empty}%

```

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

```

394 \bbl@ifshorthand{'}%
395   {\PassOptionsToPackage{activeacute}{babel}}{}
396 \bbl@ifshorthand{`}%
397   {\PassOptionsToPackage{activegrave}{babel}}{}
398 \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.

```

399 \ifx\bbl@opt@headfoot\@nnil\else
400   \g@addto@macro\@resetactivechars{%
401     \set@typeset@protect
402     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
403     \let\protect\noexpand}
404 \fi

```

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

```

405 \ifx\bbl@opt@safe\@undefined
406   \def\bbl@opt@safe{BR}
407   % \let\bbl@opt@safe\@empty % Pending of \cite
408 \fi

```

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

```

409 \bbl@trace{Defining IfBabelLayout}
410 \ifx\bbl@opt@layout\@nnil
411   \newcommand\IfBabelLayout[3]{#3}%
412 \else
413   \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
414     \in@{,layout,}{, #1,}%
415     \ifin@
416       \def\bbl@opt@layout{#2}%
417       \bbl@replace\bbl@opt@layout{ }{.}%
418     \fi}
419 \newcommand\IfBabelLayout[1]{%
420   \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
421   \ifin@
422     \expandafter\@firstoftwo
423   \else
424     \expandafter\@secondoftwo
425   \fi}
426 \fi
427 </package>
428 <*core>

```

### 3.6 Plain: babel.def (start)

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*.

```

429 \ifx\ldf@quit\@undefined\else
430 \endinput\fi % Same line!
431 <@Make sure ProvidesFile is defined@>
432 \ProvidesFile{babel.def}[<@date@> v<@version@> Babel common definitions]
433 \ifx\AtBeginDocument\@undefined %^^A TODO. change test.
434 <@Emulate LaTeX@>
435 \fi
436 <@Basic macros@>

```

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

```

437 </core>

```

## 4 babel.sty and babel.def (common)

```

438 <*package | core>
439 \def\bb@version{<@version@>}
440 \def\bb@date{<@date@>}
441 <@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.

```

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

```

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

```

457 \def\bb@fixname#1{%
458   \begingroup
459   \def\bb@tempe{l@}%
460   \edef\bb@tempd{\noexpand\@ifundefined{\noexpand\bb@tempe#1}}%
461   \bb@tempd
462     {\lowercase\expandafter{\bb@tempd}%
463     {\uppercase\expandafter{\bb@tempd}%
464     \@empty
465     {\edef\bb@tempd{\def\noexpand#1{#1}}%
466     {\uppercase\expandafter{\bb@tempd}}}%
467     {\edef\bb@tempd{\def\noexpand#1{#1}}%
468     {\lowercase\expandafter{\bb@tempd}}}%
469     \@empty
470   \edef\bb@tempd{\endgroup\def\noexpand#1{#1}}%
471   \bb@tempd
472   \bb@exp{\bb@usehooks{language}{\language}{#1}}
473 \def\bb@iflanguage#1{%
474   \@ifundefined{l@#1}{\@nolannerr{#1}\@gobble}\@firstofone}

```

After a name has been ‘fixed’, the selectors will try to load the language. If even the fixed name is not defined, will load it on the fly, either based on its name, or if activated, its BCP47 code.

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

```

475 \def\bbl@bcpcase#1#2#3#4\@#5{%
476   \ifx\@empty#3%
477     \uppercase{\def#5{#1#2}}%
478   \else
479     \uppercase{\def#5{#1}}%
480     \lowercase{\edef#5{#5#2#3#4}}%
481   \fi}
482 \def\bbl@bcpllookup#1-#2-#3-#4\@#5{%
483   \let\bbl@bcp\relax
484   \lowercase{\def\bbl@tempa{#1}}%
485   \ifx\@empty#2%
486     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
487   \else\ifx\@empty#3%
488     \bbl@bcpcase#2\@empty\@empty\@#5\bbl@tempb
489     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
490       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
491     {}%
492   \ifx\bbl@bcp\relax
493     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
494   \fi
495   \else
496     \bbl@bcpcase#2\@empty\@empty\@#5\bbl@tempb
497     \bbl@bcpcase#3\@empty\@empty\@#5\bbl@tempc
498     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
499       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
500     {}%
501   \ifx\bbl@bcp\relax
502     \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
503       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
504     {}%
505   \fi
506   \ifx\bbl@bcp\relax
507     \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
508       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
509     {}%
510   \fi
511   \ifx\bbl@bcp\relax
512     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
513   \fi
514 \fi\fi}
515 \let\bbl@initoload\relax
516 \end{package}
517 \begin{package}
518 \newif\ifbbl@bcppallowed
519 \bbl@bcppallowedfalse
520 \def\bbl@provide@locale{%
521   \ifx\babelprovide\@undefined
522     \bbl@error{base-on-the-fly}{}}}%
523 \fi
524 \let\bbl@auxname\language\language % Still necessary. %^A TODO
525 \bbl@ifunset{bbl@bcp@map@language}{% Move uplevel??
526   {\edef\language{\@nameuse{bbl@bcp@map@language}}}%
527 \ifbbl@bcppallowed
528   \expandafter\ifx\csname date\language\endcsname\relax
529     \expandafter
530     \bbl@bcpllookup\language-\@empty-\@empty-\@empty\@#5
531     \ifx\bbl@bcp\relax\else % Returned by \bbl@bcpllookup
532       \edef\language{\bbl@bcp@prefix\bbl@bcp}%

```

```

533 \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
534 \expandafter\ifx\csname date\language\endcsname\relax
535 \let\bbl@initoload\bbl@bcp
536 \bbl@exp{\\\babelprovide[\bbl@autoload@bcptoptions]{\language}}%
537 \let\bbl@initoload\relax
538 \fi
539 \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
540 \fi
541 \fi
542 \fi
543 \expandafter\ifx\csname date\language\endcsname\relax
544 \IfFileExists{babel-\language.tex}%
545 {\bbl@exp{\\\babelprovide[\bbl@autoload@options]{\language}}}%
546 {}%
547 \fi}
548 </package>
549 <*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.

```

550 \def\iflanguage#1{%
551 \bbl@iflanguage{#1}{%
552 \ifnum\csname l@#1\endcsname=\language
553 \expandafter\@firstoftwo
554 \else
555 \expandafter\@secondoftwo
556 \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.

```

557 \let\bbl@select@type\z@
558 \edef\selectlanguage{%
559 \noexpand\protect
560 \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`.

```

561 \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.

```

562 \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.

```
563 \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**

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

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

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

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

```
576 \def\bbl@pop@lang#1+#2\@{%
577   \edef\language{#1}%
578   \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).

```
579 \let\bbl@ifrestoring\@secondoftwo
580 \def\bbl@pop@language{%
581   \expandafter\bbl@pop@lang\bbl@language@stack\@
582   \let\bbl@ifrestoring\@firstoftwo
583   \expandafter\bbl@set@language\expandafter{\language}%
584   \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).

```
585 \chardef\localeid\z@
586 \def\bbl@id@last{0} % No real need for a new counter
587 \def\bbl@id@assign{%
588   \bbl@ifunset{\bbl@id@\language}%
589   {\count@\bbl@id@last\relax
590    \advance\count@\@ne
591    \bbl@csarg\chardef{id@\language}\count@
592    \edef\bbl@id@last{\the\count@}%
593    \ifcase\bbl@engine\or
594      \directlua{
```

```

595     Babel = Babel or {}
596     Babel.locale_props = Babel.locale_props or {}
597     Babel.locale_props[\bbl@id@last] = {}
598     Babel.locale_props[\bbl@id@last].name = '\language'
599   }%
600   \fi}%
601   {}%
602   \chardef\localeid\bbl@c{l{id@}}

```

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

```

603 \expandafter\def\csname selectlanguage \endcsname#1{%
604   \ifnum\bbl@hymapsel=\@ccclv\let\bbl@hymapsel\tw@\fi
605   \bbl@push@language
606   \aftergroup\bbl@pop@language
607   \bbl@set@language{#1}}
608 \let\endselectlanguage\relax

```

**\bbl@set@language** The macro `\bbl@set@language` takes care of switching the language environment *and* of writing entries on the auxiliary files. For historical reasons, language names can be either language of `\language`. To catch either form a trick is used, but unfortunately as a side effect the catcodes of letters in `\language` are messed up. This is a bug, but preserved for backwards compatibility. The list of auxiliary files can be extended by redefining `\BabelContentsFiles`, but make sure they are loaded inside a group (as `aux`, `toc`, `lof`, and `lot` do) or the last language of the document will remain active afterwards. We also write a command to change the current language in the auxiliary files. `\bbl@savelastskip` is used to deal with skips before the write whatsit (as suggested by U Fischer). Adapted from `hyperref`, but it might fail, so I'll consider it a temporary hack, while I study other options (the ideal, but very likely unfeasible except perhaps in `luatex`, is to avoid the `\write` altogether when not needed).

```

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

```

```

642 \bbl@exp{\scantokens{\def\\localename{\language}\noexpand}\relax}%
643 \fi
644 %^^A TODO. name@map must be here?
645 \bbl@provide@locale
646 \bbl@iflanguage\language\language{%
647 \let\bbl@select@type\z@
648 \expandafter\bbl@switch\expandafter{\language}}
649 \def\babel@aux#1#2{%
650 \select@language{#1}%
651 \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
652 \writefile{##1}{\babel@toc{#1}{#2}\relax}}}%^^A TODO - plain?
653 \def\babel@toc#1#2{%
654 \select@language{#1}}

```

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

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

Then we have to *redefine* `\originalTeX` to compensate for the things that have been activated. To save memory space for the macro definition of `\originalTeX`, we construct the control sequence name for the `\noextras{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`.

```

655 \newif\ifbbl@usedategroup
656 \let\bbl@savextras\@empty
657 \def\bbl@switch#1{% from select@, foreign@
658 % make sure there is info for the language if so requested
659 \bbl@ensureinfo{#1}%
660 % restore
661 \originalTeX
662 \expandafter\def\expandafter\originalTeX\expandafter{%
663 \csname noextras#1\endcsname
664 \let\originalTeX\@empty
665 \babel@beginsave}%
666 \bbl@usehooks{afterreset}{}%
667 \languageshorthands{none}%
668 % set the locale id
669 \bbl@id@assign
670 % switch captions, date
671 \bbl@bsphack
672 \ifcase\bbl@select@type
673 \csname captions#1\endcsname\relax
674 \csname date#1\endcsname\relax
675 \else
676 \bbl@xin@{,captions,}{,\bbl@select@opts,}%
677 \ifin@
678 \csname captions#1\endcsname\relax
679 \fi
680 \bbl@xin@{,date,}{,\bbl@select@opts,}%
681 \ifin@ % if \foreign... within \<language>date
682 \csname date#1\endcsname\relax
683 \fi
684 \fi
685 \bbl@esphack
686 % switch extras
687 \csname bbl@preextras@#1\endcsname
688 \bbl@usehooks{beforeextras}{}%
689 \csname extras#1\endcsname\relax

```

```

690 \bbl@usehooks{afterextras}{}%
691 % > babel-ensure
692 % > babel-sh-<short>
693 % > babel-bidi
694 % > babel-fontspec
695 \let\bbl@savextras\@empty
696 % hyphenation - case mapping
697 \ifcase\bbl@opt@hyphenmap\or
698 \def\BabelLower##1##2{\lccode##1=##2\relax}%
699 \ifnum\bbl@hymap>4\else
700 \csname\language @bbl@hyphenmap\endcsname
701 \fi
702 \chardef\bbl@opt@hyphenmap\z@
703 \else
704 \ifnum\bbl@hymap>\bbl@opt@hyphenmap\else
705 \csname\language @bbl@hyphenmap\endcsname
706 \fi
707 \fi
708 \let\bbl@hymap\@cclv
709 % hyphenation - select rules
710 \ifnum\csname l@\language\endcsname=\l@unhyphenated
711 \edef\bbl@tempa{u}%
712 \else
713 \edef\bbl@tempa{\bbl@c{lnbrk}}%
714 \fi
715 % linebreaking - handle u, e, k (v in the future)
716 \bbl@xin@{/u}{/\bbl@tempa}%
717 \ifin@else\bbl@xin@{/e}{/\bbl@tempa}\fi % elongated forms
718 \ifin@else\bbl@xin@{/k}{/\bbl@tempa}\fi % only kashida
719 \ifin@else\bbl@xin@{/p}{/\bbl@tempa}\fi % padding (eg, Tibetan)
720 \ifin@else\bbl@xin@{/v}{/\bbl@tempa}\fi % variable font
721 % hyphenation - save mins
722 \babel@savevariable\lefthyphenmin
723 \babel@savevariable\righthyphenmin
724 \ifnum\bbl@engine=\@ne
725 \babel@savevariable\hyphenationmin
726 \fi
727 \ifin@
728 % unhyphenated/kashida/elongated/padding = allow stretching
729 \language\l@unhyphenated
730 \babel@savevariable\emergencystretch
731 \emergencystretch\maxdimen
732 \babel@savevariable\hbadness
733 \hbadness\@M
734 \else
735 % other = select patterns
736 \bbl@patterns{#1}%
737 \fi
738 % hyphenation - set mins
739 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
740 \set@hyphenmins\tw@\thr@@\relax
741 \@nameuse{bbl@hyphenmins@}%
742 \else
743 \expandafter\expandafter\expandafter\set@hyphenmins
744 \csname #1hyphenmins\endcsname\relax
745 \fi
746 \@nameuse{bbl@hyphenmins@}%
747 \@nameuse{bbl@hyphenmins@\language}%
748 \@nameuse{bbl@hyphenatmin@}%
749 \@nameuse{bbl@hyphenatmin@\language}%
750 \let\bbl@selectortname\@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.

```
751 \long\def\otherlanguage#1{%
752   \def\bbl@selectorname{other}%
753   \ifnum\bbl@hymapset=\@cclv\let\bbl@hymapset\thr@@\fi
754   \csname selectlanguage \endcsname{#1}%
755   \ignorespaces}
```

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

```
756 \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`.

```
757 \expandafter\def\csname otherlanguage*\endcsname{%
758   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
759 \def\bbl@otherlanguage@s[#1]#2{%
760   \def\bbl@selectorname{other*}%
761   \ifnum\bbl@hymapset=\@cclv\chardef\bbl@hymapset4\relax\fi
762   \def\bbl@select@opts{#1}%
763   \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”.

```
764 \expandafter\let\csname endotherlanguage*\endcsname\relax
```

**foreignlanguage** 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.

```
765 \providecommand\bbl@beforeforeign{}
766 \edef\foreignlanguage{%
767   \noexpand\protect
768   \expandafter\noexpand\csname foreignlanguage \endcsname}
769 \expandafter\def\csname foreignlanguage \endcsname{%
770   \@ifstar\bbl@foreign@s\bbl@foreign@x}
771 \providecommand\bbl@foreign@x[3][]{%
772   \begingroup
773     \def\bbl@selectorname{foreign}%
774     \def\bbl@select@opts{#1}%
775     \let\BabelText\@firstofone
776     \bbl@beforeforeign
777     \foreign@language{#2}%
778     \bbl@usehooks{foreign}{}%
779     \BabelText{#3}% Now in horizontal mode!
```

```

780 \endgroup}
781 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
782 \begingroup
783 {\par}%
784 \def\bbl@selectorname{foreign*}%
785 \let\bbl@select@opts\@empty
786 \let\BabelText\@firstofone
787 \foreign@language{#1}%
788 \bbl@usehooks{foreign*}{}%
789 \bbl@dirparastext
790 \BabelText{#2}% Still in vertical mode!
791 {\par}%
792 \endgroup}
793 \providecommand\BabelWrapText[1]{%
794 \def\bbl@tempa{\def\BabelText###1}%
795 \expandafter\bbl@tempa\expandafter{\BabelText{#1}}}

```

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

```

796 \def\foreign@language#1{%
797 % set name
798 \edef\languagename{#1}%
799 \ifbbl@usedategroup
800 \bbl@add\bbl@select@opts{,date,}%
801 \bbl@usedategroupfalse
802 \fi
803 \bbl@fixname\languagename
804 \let\localename\languagename
805 % TODO. name@map here?
806 \bbl@provide@locale
807 \bbl@iflanguage\languagename{%
808 \let\bbl@select@type\@ne
809 \expandafter\bbl@switch\expandafter{\languagename}}

```

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

```

810 \def\IfBabelSelectorTF#1{%
811 \bbl@xin@{\bbl@selectorname,}{,\zap@space#1 \@empty,}%
812 \ifin@
813 \expandafter\@firstoftwo
814 \else
815 \expandafter\@secondoftwo
816 \fi}

```

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

It also sets hyphenation exceptions, but only once, because they are global (here language \lccode's has been set, too). \bbl@hyphenation@ is set to relax until the very first \babelhyphenation, so do nothing with this value. If the exceptions for a language (by its number, not its name, so that :ENC is taken into account) has been set, then use \hyphenation with both global and language exceptions and empty the latter to mark they must not be set again.

```

817 \let\bbl@hyphlist\@empty
818 \let\bbl@hyphenation@\relax
819 \let\bbl@pttnlist\@empty
820 \let\bbl@patterns@\relax
821 \let\bbl@hymapsel=\@cclv
822 \def\bbl@patterns#1{%
823 \language=\expandafter\ifx\csname l@#1:f@encoding\endcsname\relax
824 \csname l@#1\endcsname
825 \edef\bbl@tempa{#1}%
826 \else

```

```

827 \csname l@#1:\f@encoding\endcsname
828 \edef\bbl@tempa{#1:\f@encoding}%
829 \fi
830 \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
831 % > luatex
832 \@ifundefined{bbl@hyphenation@}{{}% Can be \relax!
833 \beginingroup
834 \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
835 \ifin@ \else
836 \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
837 \hyphenation{%
838 \bbl@hyphenation@
839 \@ifundefined{bbl@hyphenation@#1}%
840 \@empty
841 {\space\csname bbl@hyphenation@#1\endcsname}}%
842 \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
843 \fi
844 \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*`.

```

845 \def\hyphenrules#1{%
846 \edef\bbl@tempf{#1}%
847 \bbl@fixname\bbl@tempf
848 \bbl@iflanguage\bbl@tempf{%
849 \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
850 \ifx\languageshorthands\@undefined\else
851 \languageshorthands{none}%
852 \fi
853 \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
854 \set@hyphenmins\tw@\thr@@\relax
855 \else
856 \expandafter\expandafter\expandafter\set@hyphenmins
857 \csname\bbl@tempf hyphenmins\endcsname\relax
858 \fi}}
859 \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.

```

860 \def\providehyphenmins#1#2{%
861 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
862 \@namedef{#1hyphenmins}{#2}%
863 \fi}

```

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

```

864 \def\set@hyphenmins#1#2{%
865 \lefthyphenmin#1\relax
866 \righthyphenmin#2\relax}

```

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

```

867 \ifx\ProvidesFile\@undefined
868 \def\ProvidesLanguage#1[#2 #3 #4]{%
869 \wlog{Language: #1 #4 #3 <#2>}%
870 }

```

```

871 \else
872   \def\ProvidesLanguage#1{%
873     \begingroup
874       \catcode\ 10 %
875       \@makeother\/%
876       \@ifnextchar[%]
877         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
878   \def\@provideslanguage#1[#2]{%
879     \wlog{Language: #1 #2}%
880     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
881     \endgroup}
882 \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`.

```
883 \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.

```
884 \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’:

```

885 \providecommand\setlocale{\bbl@error{not-yet-available}}{}{}
886 \let\uselocale\setlocale
887 \let\locale\setlocale
888 \let\selectlocale\setlocale
889 \let\textlocale\setlocale
890 \let\textlanguage\setlocale
891 \let\languagetext\setlocale

```

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

```

892 \bbl@trace{Defining babelensure}
893 \newcommand\babelensure[2][]{%
894   \AddBabelHook{babel-ensure}{afterextras}{%
895     \ifcase\bbl@select@type
896       \bbl@c{l}{e}%
897     \fi}%
898   \begingroup
899     \let\bbl@ens@include\@empty
900     \let\bbl@ens@exclude\@empty
901     \def\bbl@ens@fontenc{\relax}%
902     \def\bbl@tempb##1{%
903       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
904     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
905     \def\bbl@tempb##1=##2\@{\@{\@namedef{\bbl@ens@##1}{##2}}}%
906     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}{%
907       \def\bbl@tempc{\bbl@ensure}%
908       \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
909         \expandafter{\bbl@ens@include}}%
910       \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
911         \expandafter{\bbl@ens@exclude}}%
912       \toks@\expandafter{\bbl@tempc}%
913       \bbl@exp{%
914         \endgroup

```



```

915 \def<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}}
916 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
917 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
918 \ifx##1\@undefined % 3.32 - Don't assume the macro exists
919 \edef##1{\noexpand\bbl@nocaption
920 {\bbl@stripslash##1}{\language\language\bbl@stripslash##1}}%
921 \fi
922 \ifx##1\@empty\else
923 \in@{##1}{#2}%
924 \ifin@ \else
925 \bbl@ifunset{\bbl@ensure@\language\language}%
926 {\bbl@exp{%
927 \\\DeclareRobustCommand\<bbl@ensure@\language\language>[1]{%
928 \\\foreignlanguage{\language\language}%
929 {\ifx\relax#3\else
930 \\\fontencoding{#3}\selectfont
931 \fi
932 #####1}}}%
933 }%
934 \toks@\expandafter{##1}%
935 \edef##1{%
936 \bbl@csarg\noexpand{ensure@\language\language}%
937 {\the\toks@}}%
938 \fi
939 \expandafter\bbl@tempb
940 \fi}%
941 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
942 \def\bbl@tempa##1{% elt for include list
943 \ifx##1\@empty\else
944 \bbl@csarg\in@{ensure@\language\language\expandafter}\expandafter{##1}%
945 \ifin@ \else
946 \bbl@tempb##1\@empty
947 \fi
948 \expandafter\bbl@tempa
949 \fi}%
950 \bbl@tempa#1\@empty}
951 \def\bbl@captionslist{%
952 \prefacename\refname\abstractname\bibname\chaptername\appendixname
953 \contentsname\listfigurename\listtablename\indexname\figurename
954 \tablename\partname\enclname\ccname\headtoname\pagename\seename
955 \alsoname\proofname\glossaryname}

```

## 4.2 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.

```

956 \bbl@trace{Short tags}
957 \def\babeltags#1{%
958 \edef\bbl@tempa{\zap@space#1 \@empty}%
959 \def\bbl@tempb##1=##2\@@{%
960 \edef\bbl@tempc{%
961 \noexpand\newcommand
962 \expandafter\noexpand\csname ##1\endcsname{%
963 \noexpand\protect
964 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
965 \noexpand\newcommand
966 \expandafter\noexpand\csname text##1\endcsname{%
967 \noexpand\foreignlanguage{##2}}}}
968 \bbl@tempc}%
969 \bbl@for\bbl@tempa\bbl@tempa{%
970 \expandafter\bbl@tempb\bbl@tempa\@@}}

```

## 4.3 Errors

### **\@nolanerr**

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

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

When the format knows about \PackageError it must be  $\text{\TeX 2}_{\epsilon}$ , 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.

```
971 \edef\bbl@nulllanguage{\string\language=0}
972 \def\bbl@nocaption{\protect\bbl@nocaption@i}
973 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
974   \global\@namedef{#2}{\textbf{?#1?}}}%
975   \@nameuse{#2}%
976   \edef\bbl@tempa{#1}%
977   \bbl@sreplace\bbl@tempa{name}{}%
978   \bbl@warning{%
979     \@backslashchar#1 not set for '\language'. Please,\\%
980     define it after the language has been loaded\\%
981     (typically in the preamble) with:\\%
982     \string\setlocalecaption{\language}{\bbl@tempa}{..}\\%
983     Feel free to contribute on github.com/latex3/babel.\\%
984     Reported}}
985 \def\bbl@tentative{\protect\bbl@tentative@i}
986 \def\bbl@tentative@i#1{%
987   \bbl@warning{%
988     Some functions for '#1' are tentative.\\%
989     They might not work as expected and their behavior\\%
990     could change in the future.\\%
991     Reported}}
992 \def\@nolanerr#1{\bbl@error{undefined-language}{#1}{}}
993 \def\@nopatterns#1{%
994   \bbl@warning
995     {No hyphenation patterns were preloaded for\\%
996     the language '#1' into the format.\\%
997     Please, configure your TeX system to add them and\\%
998     rebuild the format. Now I will use the patterns\\%
999     preloaded for \bbl@nulllanguage\space instead}}
1000 \let\bbl@usehooks\@gobbletwo
1001 \ifx\bbl@onlyswitch\@empty\endinput\fi
1002 % Here ended switch.def
```

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

```
1003 \ifx\directlua\@undefined\else
1004   \ifx\bbl@luapatterns\@undefined
1005     \input luababel.def
1006   \fi
1007 \fi
1008 \bbl@trace{Compatibility with language.def}
1009 \ifx\bbl@languages\@undefined
1010   \ifx\directlua\@undefined
1011     \openin1 = language.def % TODO. Remove hardcoded number
1012     \ifeof1
1013       \closein1
1014       \message{I couldn't find the file language.def}
1015     \else
1016       \closein1
```

```

1017 \begingroup
1018 \def\addlanguage#1#2#3#4#5{%
1019 \expandafter\ifx\csname lang@#1\endcsname\relax\else
1020 \global\expandafter\let\csname l@#1\expandafter\endcsname
1021 \csname lang@#1\endcsname
1022 \fi}%
1023 \def\uselanguage#1{%
1024 \input language.def
1025 \endgroup
1026 \fi
1027 \fi
1028 \chardef\l@english\z@
1029 \fi

```

**\addto** It takes two arguments, a *<control sequence>* and  $\TeX$ -code to be added to the *<control sequence>*.

If the *<control sequence>* has not been defined before it is defined now. The control sequence could also expand to `\relax`, in which case a circular definition results. The net result is a stack overflow. Note there is an inconsistency, because the assignment in the last branch is global.

```

1030 \def\addto#1#2{%
1031 \ifx#1\@undefined
1032 \def#1{#2}%
1033 \else
1034 \ifx#1\relax
1035 \def#1{#2}%
1036 \else
1037 {\toks@\expandafter{#1#2}%
1038 \xdef#1{the\toks@}}%
1039 \fi
1040 \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.

```

1041 \def\bbl@withactive#1#2{%
1042 \begingroup
1043 \lccode`~=#2\relax
1044 \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  $\LaTeX$  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`.

```

1045 \def\bbl@redefine#1{%
1046 \edef\bbl@tempa{\bbl@stripslash#1}%
1047 \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1048 \expandafter\def\csname\bbl@tempa\endcsname}
1049 \@onlypreamble\bbl@redefine

```

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

```

1050 \def\bbl@redefine@long#1{%
1051 \edef\bbl@tempa{\bbl@stripslash#1}%
1052 \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1053 \long\expandafter\def\csname\bbl@tempa\endcsname}
1054 \@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_`.

```

1055 \def\bbl@redefineroobust#1{%
1056   \edef\bbl@tempa{\bbl@stripslash#1}%
1057   \bbl@ifunset{\bbl@tempa\space}%
1058     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1059       \bbl@exp{\def\\#1{\protect\<\bbl@tempa\space>}}}%
1060     {\bbl@exp{\let\<org@\bbl@tempa\<\bbl@tempa\space>}}}%
1061     \@namedef{\bbl@tempa\space}}
1062 \@onlypreamble\bbl@redefineroobust

```

## 4.4 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.

```

1063 \bbl@trace{Hooks}
1064 \newcommand\AddBabelHook[3][]{%
1065   \bbl@ifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}{%
1066     \def\bbl@tempa##1,##3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1067     \expandafter\bbl@tempa\bbl@evargs,##3=,\@empty
1068     \bbl@ifunset{\bbl@ev@#2@#3@#1}%
1069       {\bbl@csarg\bbl@add{\ev@#3@#1}{\bbl@elth{#2}}}%
1070       {\bbl@csarg\let{\ev@#2@#3@#1}\relax}%
1071     \bbl@csarg\newcommand{\ev@#2@#3@#1}{\bbl@tempb}}
1072 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{\hk@#1}\@firstofone}
1073 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{\hk@#1}\@gobble}
1074 \def\bbl@usehooks{\bbl@usehooks@lang\language}
1075 \def\bbl@usehooks@lang#1#2#3{% Test for Plain
1076   \ifx\UseHook\@undefined\else\UseHook{babel/*/#2}\fi
1077   \def\bbl@elth##1{%
1078     \bbl@cs{\hk@##1}{\bbl@cs{\ev@##1@#2@#3}}%
1079     \bbl@cs{\ev@#2@}}%
1080   \ifx\language\@undefined\else % Test required for Plain (?)
1081     \ifx\UseHook\@undefined\else\UseHook{babel/#1/#2}\fi
1082     \def\bbl@elth##1{%
1083       \bbl@cs{\hk@##1}{\bbl@cs{\ev@##1@#2@#1@#3}}%
1084       \bbl@cs{\ev@#2@#1}}%
1085   \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).

```

1086 \def\bbl@evargs{,% <- don't delete this comma
1087   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1088   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1089   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1090   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1091   beforestart=0,language=2,begindocument=1}
1092 \ifx\NewHook\@undefined\else % Test for Plain (?)
1093   \def\bbl@tempa#1=#2\@{\NewHook{babel/#1}}
1094   \bbl@foreach\bbl@evargs{\bbl@tempa#1\@}
1095 \fi

```

## 4.5 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`.

```

1096 \bbl@trace{Macros for setting language files up}
1097 \def\bbl@ldfinit{%
1098   \let\bbl@screset\@empty
1099   \let\BabelStrings\bbl@opt@string
1100   \let\BabelOptions\@empty
1101   \let\BabelLanguages\relax
1102   \ifx\originalTeX\@undefined
1103     \let\originalTeX\@empty
1104   \else
1105     \originalTeX
1106   \fi}
1107 \def\LdfInit#1#2{%
1108   \chardef\atcatcode=\catcode`\@
1109   \catcode`\@=11\relax
1110   \chardef\eqcatcode=\catcode`\=
1111   \catcode`\==12\relax
1112   \expandafter\if\expandafter\@backslashchar
1113     \expandafter\@car\string#2\@nil
1114   \ifx#2\@undefined\else
1115     \ldf@quit{#1}%
1116   \fi
1117 \else
1118   \expandafter\ifx\csname#2\endcsname\relax\else
1119     \ldf@quit{#1}%
1120   \fi
1121 \fi
1122 \bbl@ldfinit}

```

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

```

1123 \def\ldf@quit#1{%
1124   \expandafter\main@language\expandafter{#1}%
1125   \catcode`\@=\atcatcode \let\atcatcode\relax
1126   \catcode`\==\eqcatcode \let\eqcatcode\relax
1127   \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.

```

1128 \def\bbl@afterldf#1{%%^A TODO. #1 is not used. Remove
1129   \bbl@afterlang
1130   \let\bbl@afterlang\relax
1131   \let\BabelModifiers\relax
1132   \let\bbl@screset\relax}%
1133 \def\ldf@finish#1{%
1134   \loadlocalcfg{#1}%
1135   \bbl@afterldf{#1}%
1136   \expandafter\main@language\expandafter{#1}%
1137   \catcode`\@=\atcatcode \let\atcatcode\relax
1138   \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  $\LaTeX$ .

```
1139 \@onlypreamble\LdfInit
1140 \@onlypreamble\ldf@quit
1141 \@onlypreamble\ldf@finish
```

## **`\main@language`**

**`\bbl@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.

```
1142 \def\main@language#1{%
1143   \def\bbl@main@language{#1}%
1144   \let\language\main@language
1145   \let\localename\bbl@main@language
1146   \let\mainlocalename\bbl@main@language
1147   \bbl@id@assign
1148   \bbl@patterns{\language}}
```

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

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

```
1149 \def\bbl@beforestart{%
1150   \def\@nolanerr##1{%
1151     \bbl@carg\chardef\l@##1}\z@
1152     \bbl@warning{Undefined language '##1' in aux.\\Reported}}%
1153   \bbl@usehooks{beforestart}{}%
1154   \global\let\bbl@beforestart\relax}
1155 \AtBeginDocument{%
1156   {\@nameuse{bbl@beforestart}}% Group!
1157   \if@filesw
1158     \providecommand\babel@aux[2]{}%
1159     \immediate\write\@mainaux{\unexpanded{%
1160       \providecommand\babel@aux[2]{\global\let\babel@toc@gobbletwo}}}%
1161     \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1162   \fi
1163   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1164 \</package | core>
1165 \< *package>
1166   \ifx\bbl@normalsf\@empty
1167     \ifnum\sfcodes\@m
1168       \let\normalsfcodes\frenchspacing
1169     \else
1170       \let\normalsfcodes\nonfrenchspacing
1171     \fi
1172   \else
1173     \let\normalsfcodes\bbl@normalsf
1174   \fi
1175 \</package>
1176 \< *package | core>
1177   \ifbbl@single % must go after the line above.
1178     \renewcommand\selectlanguage[1]{}%
1179     \renewcommand\foreignlanguage[2]{#2}%
1180     \global\let\babel@aux\@gobbletwo % Also as flag
1181   \fi}
1182 \</package | core>
1183 \< *package>
1184 \AddToHook{begindocument/before}{%
1185   \let\bbl@normalsf\normalsfcodes
1186   \let\normalsfcodes\relax} % Hack, to delay the setting
1187 \</package>%
```

```

1188 <{*package | core}>
1189 \ifcase\bbl@engine\or
1190   \AtBeginDocument{\pagedir\bodydir} %^^A TODO - a better place
1191 \fi

```

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

```

1192 \def\select@language@x#1{%
1193   \ifcase\bbl@select@type
1194     \bbl@ifsamestring\language#1\{\select@language#1}%
1195   \else
1196     \select@language#1%
1197 \fi}

```

## 4.6 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.

```

1198 \bbl@trace{Shorhands}
1199 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1200   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1201   \bbl@ifunset{@sanitize}\{\bbl@add\@sanitize{\@makeother#1}%
1202   \ifx\nfss@catcodes\undefined\else % TODO - same for above
1203     \begingroup
1204       \catcode`#1\active
1205       \nfss@catcodes
1206       \ifnum\catcode`#1=\active
1207         \endgroup
1208       \bbl@add\nfss@catcodes{\@makeother#1}%
1209     \else
1210       \endgroup
1211   \fi
1212 \fi}

```

**\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).

```

1213 \def\bbl@active@def#1#2#3#4{%
1214   \@namedef{#3#1}{%
1215     \expandafter\ifx\csname#2@sh#1@endcsname\relax
1216       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1217     \else
1218       \bbl@afterfi\csname#2@sh#1@endcsname
1219     \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.

```

1220 \long\@namedef{#3@arg#1}##1{%
1221   \expandafter\ifx\csname#2@sh@#1@\string##1\endcsname\relax
1222     \bbl@afterelse\csname#4#1\endcsname##1%
1223   \else
1224     \bbl@afterfi\csname#2@sh@#1@\string##1\endcsname
1225   \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.

```

1226 \def\initiate@active@char#1{%
1227   \bbl@ifunset{active@char\string#1}%
1228   {\bbl@withactive
1229     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1230   {}}

```

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).

```

1231 \def\@initiate@active@char#1#2#3{%
1232   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1233   \ifx#1\@undefined
1234     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1235   \else
1236     \bbl@csarg\let{oridef@#2}#1%
1237     \bbl@csarg\edef{oridef@#2}{%
1238       \let\noexpand#1%
1239       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1240   \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*").

```

1241 \ifx#1#3\relax
1242   \expandafter\let\csname normal@char#2\endcsname#3%
1243 \else
1244   \bbl@info{Making #2 an active character}%
1245   \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1246   \@namedef{normal@char#2}{%
1247     \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
1248   \else
1249     \@namedef{normal@char#2}{#3}%
1250   \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).

```

1251 \bbl@restoreactive{#2}%
1252 \AtBeginDocument{%
1253   \catcode`#2\active
1254   \if@filesw
1255     \immediate\write\@mainaux{\catcode`\string#2\active}%
1256   \fi}%
1257 \expandafter\bbl@add@special\csname#2\endcsname
1258 \catcode`#2\active
1259 \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⟩`).

```

1260 \let\bbl@tempa\@firstoftwo
1261 \if\string^#2%
1262   \def\bbl@tempa{\noexpand\textormath}%
1263 \else
1264   \ifx\bbl@mathnormal\@undefined\else
1265     \let\bbl@tempa\bbl@mathnormal
1266   \fi
1267 \fi
1268 \expandafter\edef\csname active@char#2\endcsname{%
1269   \bbl@tempa
1270   {\noexpand\if@safe@actives
1271     \noexpand\expandafter
1272     \expandafter\noexpand\csname normal@char#2\endcsname
1273     \noexpand\else
1274       \noexpand\expandafter
1275       \expandafter\noexpand\csname bbl@doactive#2\endcsname
1276       \noexpand\fi}%
1277   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1278 \bbl@csarg\edef{doactive#2}{%
1279   \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

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

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

```

1280 \bbl@csarg\edef{active@#2}{%
1281   \noexpand\active@prefix\noexpand#1%
1282   \expandafter\noexpand\csname active@char#2\endcsname}%
1283 \bbl@csarg\edef{normal@#2}{%
1284   \noexpand\active@prefix\noexpand#1%
1285   \expandafter\noexpand\csname normal@char#2\endcsname}%
1286 \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.

```

1287 \bbl@active@def#2\user@group{user@active}{language@active}%
1288 \bbl@active@def#2\language@group{language@active}{system@active}%
1289 \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  $\mathrm{T}_\mathrm{E}\mathrm{X}$  would see `\protect'\protect'`. To prevent this from happening a couple of shorthand needs to be defined at user level.

```

1290 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1291   {\expandafter\noexpand\csname normal@char#2\endcsname}%
1292 \expandafter\edef\csname\user@group @sh#2@\string\protect@\endcsname
1293   {\expandafter\noexpand\csname user@active#2\endcsname}%

```

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

```

1294 \if\string'#2%
1295   \let\prim@s\bbl@prim@s

```

```

1296 \let\active@math@prime#1%
1297 \fi
1298 \bbl@usehooks{initiateactive}{#1}{#2}{#3}}

```

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

```

1299 <<{*More package options}>> ≡
1300 \DeclareOption{math=active}{}
1301 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1302 <</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*.

```

1303 \ifpackagewith{babel}{KeepShorthandsActive}%
1304 {\let\bbl@restoreactive@gobble}%
1305 {\def\bbl@restoreactive#1{%
1306   \bbl@exp{%
1307     \\AfterBabelLanguage\\CurrentOption
1308     {\catcode`#1=\the\catcode`#1\relax}%
1309     \\AtEndOfPackage
1310     {\catcode`#1=\the\catcode`#1\relax}}}%
1311   \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.

```

1312 \def\bbl@sh@select#1#2{%
1313   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1314     \bbl@afterelse\bbl@scndcs
1315   \else
1316     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1317   \fi}

```

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

```

1318 \begingroup
1319 \bbl@ifunset{ifincsname}%%^A Ugly. Correct? Only Plain?
1320 {\gdef\active@prefix#1{%
1321   \ifx\protect\@typeset@protect
1322   \else
1323     \ifx\protect\@unexpandable@protect
1324       \noexpand#1%
1325     \else
1326       \protect#1%
1327     \fi
1328   \expandafter\@gobble
1329   \fi}}
1330 {\gdef\active@prefix#1{%
1331   \ifincsname
1332     \string#1%
1333     \expandafter\@gobble
1334   \else
1335     \ifx\protect\@typeset@protect
1336     \else
1337       \ifx\protect\@unexpandable@protect
1338         \noexpand#1%

```

```

1339         \else
1340         \protect#1%
1341         \fi
1342         \expandafter\expandafter\expandafter\@gobble
1343         \fi
1344     \fi}}
1345 \endgroup

```

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

```

1346 \newif\if@safe@actives
1347 \@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.

```

1348 \def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}

```

**\bbl@activate**

**\bbl@deactivate** 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.

```

1349 \chardef\bbl@activated\z@
1350 \def\bbl@activate#1{%
1351   \chardef\bbl@activated\@ne
1352   \bbl@withactive{\expandafter\let\expandafter}#1%
1353   \csname bbl@active@\string#1\endcsname}
1354 \def\bbl@deactivate#1{%
1355   \chardef\bbl@activated\tw@
1356   \bbl@withactive{\expandafter\let\expandafter}#1%
1357   \csname bbl@normal@\string#1\endcsname}

```

**\bbl@firstcs**

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

```

1358 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1359 \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 T<sub>E</sub>X code in text mode, (2) the string for hyperref, (3) the T<sub>E</sub>X 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.

```

1360 \def\babel@texpdf#1#2#3#4{%
1361   \ifx\texorpdfstring\undefined
1362     \textormath{#1}{#3}%
1363   \else
1364     \texorpdfstring{\textormath{#1}{#3}}{#2}%

```

```

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

```

1394 \def\textormath{%
1395 \ifmmode
1396 \expandafter\@secondoftwo
1397 \else
1398 \expandafter\@firstoftwo
1399 \fi}

```

**\user@group**

**\language@group**

**\system@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 group ‘english’ and have a system group called ‘system’.

```

1400 \def\user@group{user}
1401 \def\language@group{english} %^^A I don't like defaults
1402 \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.

```

1403 \def\useshorthands{%
1404 \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}
1405 \def\bbl@usesh@s#1{%
1406 \bbl@usesh@x
1407 {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1408 {#1}}
1409 \def\bbl@usesh@x#1#2{%
1410 \bbl@ifshorthand{#2}%

```

```

1411 {\def\user@group{user}%
1412 \initiate@active@char{#2}%
1413 #1%
1414 \bbl@activate{#2}}%
1415 {\bbl@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 `\bbl@set@user@generic`); we make also sure `{}` and `\protect` are taken into account in this new top level.

```

1416 \def\user@language@group{user@\language@group}
1417 \def\bbl@set@user@generic#1#2{%
1418 \bbl@ifunset{user@generic@active#1}%
1419 {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1420 \bbl@active@def#1\user@group{user@generic@active}{language@active}%
1421 \expandafter\edef\csname#2@sh@#1@\endcsname{%
1422 \expandafter\noexpand\csname normal@char#1\endcsname}%
1423 \expandafter\edef\csname#2@sh@#1@string\protect\endcsname{%
1424 \expandafter\noexpand\csname user@active#1\endcsname}}%
1425 \@empty}
1426 \newcommand\defineshorthand[3][user]{%
1427 \edef\bbl@tempa{\zap@space#1 \@empty}%
1428 \bbl@for\bbl@tempb\bbl@tempa{%
1429 \if*\expandafter\@car\bbl@tempb\@nil
1430 \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1431 \@expandtwoargs
1432 \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1433 \fi
1434 \declare@shorthand{\bbl@tempb}{#2}{#3}}

```

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

```

1435 \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`.

```

1436 \def\aliasshorthand#1#2{%
1437 \bbl@ifshorthand{#2}%
1438 {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1439 \ifx\document\@notprerr
1440 \@notshorthand{#2}%
1441 \else
1442 \initiate@active@char{#2}%
1443 \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1444 \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1445 \bbl@activate{#2}%
1446 \fi
1447 \fi}%
1448 {\bbl@error{shorthand-is-off}}{#2}{}}

```

**\@notshorthand**

```

1449 \def\@notshorthand#1{\bbl@error{not-a-shorthand}{#1}{}}

```

**\shorthandon**

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

```
1450 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1451 \DeclareRobustCommand*\shorthandoff{%
1452   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1453 \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.

```
1454 \def\bbl@switch@sh#1#2{%
1455   \ifx#2\@nnil\else
1456     \bbl@ifunset{\bbl@active@\string#2}%
1457     {\bbl@error{not-a-shorthand-b}{\string#2}}}%
1458     {\ifcase#1%   off, on, off*
1459       \catcode`#2\relax
1460       \or
1461       \catcode`#2\active
1462       \bbl@ifunset{\bbl@shdef@\string#2}%
1463       {}%
1464       {\bbl@withactive{\expandafter\let\expandafter}#2%
1465         \csname bbl@shdef@\string#2\endcsname
1466         \bbl@csarg\let{\shdef@\string#2}\relax}%
1467       \ifcase\bbl@activated\or
1468       \bbl@activate{#2}%
1469       \else
1470       \bbl@deactivate{#2}%
1471       \fi
1472       \or
1473       \bbl@ifunset{\bbl@shdef@\string#2}%
1474       {\bbl@withactive{\bbl@csarg\let{\shdef@\string#2}}#2}%
1475       {}%
1476       \csname bbl@oricat@\string#2\endcsname
1477       \csname bbl@oridef@\string#2\endcsname
1478       \fi}%
1479   \bbl@afterfi\bbl@switch@sh#1%
1480   \fi}
```

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

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

```

1499   \fi}
1500   \let\bbl@s@activate\bbl@activate
1501   \def\bbl@activate#1{%
1502     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1503   \let\bbl@s@deactivate\bbl@deactivate
1504   \def\bbl@deactivate#1{%
1505     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1506 \fi

```

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

```

1507 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{\bbl@active@\string#1}{#3}{#2}}

```

## **\bbl@prim@s**

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

```

1508 \def\bbl@prim@s{%
1509   \prime\futurelet\@let@token\bbl@prim@s}
1510 \def\bbl@if@primes#1#2{%
1511   \ifx#1\@let@token
1512     \expandafter\@firstoftwo
1513   \else\ifx#2\@let@token
1514     \bbl@afterelse\expandafter\@firstoftwo
1515   \else
1516     \bbl@afterfi\expandafter\@secondoftwo
1517   \fi\fi}
1518 \begingroup
1519   \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
1520   \catcode`\'=12 \catcode`\"=\active \lccode`\"=\'
1521   \lowercase{%
1522     \gdef\bbl@prim@s{%
1523       \bbl@if@primes"'"%
1524       \pr@@@s
1525       {\bbl@if@primes*\^*\pr@@@t\egroup}}
1526 \endgroup

```

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

```

1527 \initiate@active@char{~}
1528 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1529 \bbl@activate{~}

```

## **\OT1dqpos**

**\T1dqpos** 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.

```

1530 \expandafter\def\csname OT1dqpos\endcsname{127}
1531 \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

```

1532 \ifx\f@encoding\undefined
1533   \def\f@encoding{OT1}
1534 \fi

```

## 4.7 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.

```
1535 \bbl@trace{Language attributes}
1536 \newcommand\languageattribute[2]{%
1537   \def\bbl@tempc{#1}%
1538   \bbl@fixname\bbl@tempc
1539   \bbl@iflanguage\bbl@tempc{%
1540     \bbl@vforeach{#2}{%
```

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

```
1541     \ifx\bbl@known@attrs\undefined
1542       \in@false
1543     \else
1544       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
1545     \fi
1546     \ifin@
1547       \bbl@warning{%
1548         You have more than once selected the attribute '##1'\%
1549         for language #1. Reported}%
1550     \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.

```
1551     \bbl@exp{%
1552       \\ \bbl@add@list\\ \bbl@known@attrs{\bbl@tempc-##1}}%
1553     \edef\bbl@tempa{\bbl@tempc-##1}%
1554     \expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes%
1555     {\csname\bbl@tempc @attr##1\endcsname}%
1556     {\@attrerr{\bbl@tempc}{##1}}%
1557     \fi}}
1558 \@onlypreamble\languageattribute
```

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

```
1559 \newcommand*{\@attrerr}[2]{%
1560   \bbl@error{unknown-attribute}{#1}{#2}{}}
```

**\bbl@declare@ttribute** This command adds the new language/attribute combination to the list of known attributes.

Then it defines a control sequence to be executed when the attribute is used in a document. The result of this should be that the macro `\extras...` for the current language is extended, otherwise the attribute will not work as its code is removed from memory at `\begin{document}`.

```
1561 \def\bbl@declare@ttribute#1#2#3{%
1562   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1563   \ifin@
1564     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1565   \fi
1566   \bbl@add@list\bbl@attributes{#1-#2}%
1567   \expandafter\def\csname#1@attr@#2\endcsname{#3}}
```

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



```

1568 \def\bbl@ifattributeset#1#2#3#4{%
1569   \ifx\bbl@known@attrs\@undefined
1570     \in@false
1571   \else
1572     \bbl@xin@{,#1-#2,}{,\bbl@known@attrs,}%
1573   \fi
1574   \ifin@
1575     \bbl@afterelse#3%
1576   \else
1577     \bbl@afterfi#4%
1578   \fi}

```

**\bbl@ifknown@ttrib** An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the  $\TeX$ -code to be executed when the attribute is known and the  $\TeX$ -code to be executed otherwise. We first assume the attribute is unknown. Then we loop over the list of known attributes, trying to find a match.

```

1579 \def\bbl@ifknown@ttrib#1#2{%
1580   \let\bbl@tempa\@secondoftwo
1581   \bbl@loopx\bbl@tempb{#2}{%
1582     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1583   \ifin@
1584     \let\bbl@tempa\@firstoftwo
1585   \else
1586     \fi}%
1587   \bbl@tempa}

```

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

```

1588 \def\bbl@clear@ttribs{%
1589   \ifx\bbl@attributes\@undefined\else
1590     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1591       \expandafter\bbl@clear@ttrib\bbl@tempa.}%
1592     \let\bbl@attributes\@undefined
1593   \fi}
1594 \def\bbl@clear@ttrib#1-#2.{%
1595   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1596 \AtBeginDocument{\bbl@clear@ttribs}

```

## 4.8 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**

**\babel@beginsave** The initialization of a new save cycle: reset the counter to zero.

```

1597 \bbl@trace{Macros for saving definitions}
1598 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

1599 \newcount\babel@savecnt
1600 \babel@beginsave

```

**\babel@save**

**\babel@savevariable** The macro `\babel@save<csname>` saves the current meaning of the control sequence `<csname>` to `\originalTeX`<sup>2</sup>. 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.

```

1601 \def\babel@save#1{%
1602   \def\bbl@tempa{,{#1,}}% Clumsy, for Plain
1603   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1604     \expandafter{\expandafter,\bbl@savextras,}}%
1605   \expandafter\in@\bbl@tempa
1606   \ifin@ \else
1607     \bbl@add\bbl@savextras{,{#1,}}%
1608     \bbl@carg\let\babel@number\babel@savecnt#1\relax
1609     \toks@\expandafter{\originalTeX\let#1=}%
1610     \bbl@exp{%
1611       \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1612     \advance\babel@savecnt@one
1613   \fi}
1614 \def\babel@savevariable#1{%
1615   \toks@\expandafter{\originalTeX #1=}%
1616   \bbl@exp{\def\\originalTeX{\the\toks@the#1\relax}}}

```

### **\bbl@frenchspacing**

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

```

1617 \def\bbl@frenchspacing{%
1618   \ifnum\the\sfcodes`.\=@m
1619     \let\bbl@nonfrenchspacing\relax
1620   \else
1621     \frenchspacing
1622     \let\bbl@nonfrenchspacing\nonfrenchspacing
1623   \fi}
1624 \let\bbl@nonfrenchspacing\nonfrenchspacing
1625 \let\bbl@elt\relax
1626 \edef\bbl@fs@chars{%
1627   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1628   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1629   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1630 \def\bbl@pre@fs{%
1631   \def\bbl@elt##1##2##3{\sfcodes`##1=\the\sfcodes`##1\relax}%
1632   \edef\bbl@save@sfcodes{\bbl@fs@chars}%
1633 \def\bbl@post@fs{%
1634   \bbl@save@sfcodes
1635   \edef\bbl@tempa{\bbl@ccl{frspc}}%
1636   \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1637   \if u\bbl@tempa % do nothing
1638   \else\if n\bbl@tempa % non french
1639     \def\bbl@elt##1##2##3{%
1640       \ifnum\sfcodes`##1=##2\relax
1641         \babel@savevariable{\sfcodes`##1}%
1642         \sfcodes`##1=##3\relax
1643       \fi}%
1644     \bbl@fs@chars
1645   \else\if y\bbl@tempa % french
1646     \def\bbl@elt##1##2##3{%

```

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

```

1647 \ifnum\sfcode`##1=##3\relax
1648 \babel@savevariable{\sfcode`##1}%
1649 \sfcode`##1=##2\relax
1650 \fi}%
1651 \bbl@fs@chars
1652 \fi\fi\fi}

```

## 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.

```

1653 \bbl@trace{Hyphens}
1654 \@onlypreamble\babelhyphenation
1655 \AtEndOfPackage{%
1656 \newcommand\babelhyphenation[2][\@empty]{%
1657 \ifx\bbl@hyphenation@\relax
1658 \let\bbl@hyphenation@\@empty
1659 \fi
1660 \ifx\bbl@hyphlist\@empty\else
1661 \bbl@warning{%
1662 You must not intermingle \string\selectlanguage\space and\\%
1663 \string\babelhyphenation\space or some exceptions will not\\%
1664 be taken into account. Reported}%
1665 \fi
1666 \ifx\@empty#1%
1667 \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1668 \else
1669 \bbl@vforeach{#1}{%
1670 \def\bbl@tempa{##1}%
1671 \bbl@fixname\bbl@tempa
1672 \bbl@iflanguage\bbl@tempa{%
1673 \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1674 \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1675 {}%
1676 {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1677 #2}}}%
1678 \fi}}

```

**\babelhyphenmins** Only  $\text{\LaTeX}$  (basically because it's defined with a  $\text{\LaTeX}$  tool).

```

1679 \ifx\NewDocumentCommand\@undefined\else
1680 \NewDocumentCommand\babelhyphenmins{sommo}{%
1681 \IfNoValueTF{#2}%
1682 {\protected@edef\bbl@hyphenmins@\set@hyphenmins{#3}{#4}}%
1683 \IfValueT{#5}{%
1684 \protected@edef\bbl@hyphenatmin@{\hyphenationmin=#5\relax}}%
1685 \IfBooleanT{#1}{%
1686 \lefthyphenmin=#3\relax
1687 \righthyphenmin=#4\relax
1688 \IfValueT{#5}{\hyphenationmin=#5\relax}}}%
1689 {\edef\bbl@tempb{\zap@space#2 \@empty}%
1690 \bbl@for\bbl@tempa\bbl@tempb{%
1691 \namedef{bbl@hyphenmins@\bbl@tempa}{\set@hyphenmins{#3}{#4}}%
1692 \IfValueT{#5}{%
1693 \@namedef{bbl@hyphenatmin@\bbl@tempa}{\hyphenationmin=#5\relax}}}%
1694 \IfBooleanT{#1}{\bbl@error{hyphenmins-args}{}}}%
1695 \fi

```

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

```
1696 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1697 \def\bbl@t@one{Tl}
1698 \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`.

```
1699 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1700 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1701 \def\bbl@hyphen{%
1702   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
1703 \def\bbl@hyphen@i#1#2{%
1704   \bbl@ifunset{\bbl@hy@#1#2\@empty}%
1705     {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1706     {\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.

```
1707 \def\bbl@usehyphen#1{%
1708   \leavevmode
1709   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1710   \nobreak\hskip\z@skip}
1711 \def\bbl@usehyphen#1{%
1712   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char:

```
1713 \def\bbl@hyphenchar{%
1714   \ifnum\hyphenchar\font=\m@ne
1715     \babelnullhyphen
1716   \else
1717     \char\hyphenchar\font
1718   \fi}
```

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

```
1719 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1720 \def\bbl@hy@@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1721 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1722 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1723 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}{}}
1724 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1725 \def\bbl@hy@repeat{%
1726   \bbl@usehyphen{%
1727     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}{}}
1728 \def\bbl@hy@@repeat{%
1729   \bbl@usehyphen{%
1730     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}{}}
1731 \def\bbl@hy@empty{\hskip\z@skip}
1732 \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.

```
1733 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bbl@allowhyphens}
```

<sup>3</sup>TeX begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

## 4.10 Multiencoding strings

The aim following commands is to provide a common interface for strings in several encodings. They also contains several hooks which can be used by luatex and xetex. The code is organized here with pseudo-guards, so we start with the basic commands.

**Tools** But first, a tool. It makes global a local variable. This is not the best solution, but it works.

```
1734 \bbl@trace{Multiencoding strings}
1735 \def\bbl@toglobal#1{\global\let#1#1}
```

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

```
1736 <<{*More package options}>> ≡
1737 \DeclareOption{nocase}{}
1738 <</More package options>>
```

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

```
1739 <<{*More package options}>> ≡
1740 \let\bbl@opt@strings\@nnil % accept strings=value
1741 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1742 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1743 \def\BabelStringsDefault{generic}
1744 <</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.

```
1745 \@onlypreamble\StartBabelCommands
1746 \def\StartBabelCommands{%
1747   \begingroup
1748   \@tempcnta="7F
1749   \def\bbl@tempa{%
1750     \ifnum\@tempcnta>"FF\else
1751       \catcode\@tempcnta=11
1752       \advance\@tempcnta\@ne
1753       \expandafter\bbl@tempa
1754     \fi}%
1755   \bbl@tempa
1756   <@Macros local to BabelCommands@>
1757   \def\bbl@provstring##1##2{%
1758     \providecommand##1{##2}%
1759     \bbl@toglobal##1}%
1760   \global\let\bbl@scafter\@empty
1761   \let\StartBabelCommands\bbl@startcmds
1762   \ifx\BabelLanguages\relax
1763     \let\BabelLanguages\CurrentOption
1764   \fi
1765   \begingroup
1766   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1767   \StartBabelCommands}
1768 \def\bbl@startcmds{%
1769   \ifx\bbl@screset\@nnil\else
1770     \bbl@usehooks{stopcommands}{}%
1771   \fi
1772   \endgroup
1773   \begingroup
1774   \ifstar
1775     {\ifx\bbl@opt@strings\@nnil
1776       \let\bbl@opt@strings\BabelStringsDefault
1777     \fi
1778     \bbl@startcmds@i}%
1779   \bbl@startcmds@i}
1780 \def\bbl@startcmds@i#1#2{%
1781   \edef\bbl@L{\zap@space#1 \@empty}%
```

```

1782 \edef\bbl@G{\zap@space#2 \@empty}%
1783 \bbl@startcmds@ii}
1784 \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.

```

1785 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1786 \let\SetString@gobbletwo
1787 \let\bbl@stringdef@gobbletwo
1788 \let\AfterBabelCommands@gobble
1789 \ifx\@empty#1%
1790 \def\bbl@sc@label{generic}%
1791 \def\bbl@encstring##1##2{%
1792 \ProvideTextCommandDefault##1{##2}%
1793 \bbl@toglobal##1%
1794 \expandafter\bbl@toglobal\csname\string?string##1\endcsname}%
1795 \let\bbl@sctest\in@true
1796 \else
1797 \let\bbl@sc@charset\space % <- zapped below
1798 \let\bbl@sc@fontenc\space % <- " "
1799 \def\bbl@tempa##1=##2\@nil{%
1800 \bbl@csarg\edef{sc\zap@space##1 \@empty}{##2 }}%
1801 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1802 \def\bbl@tempa##1 ##2{% space -> comma
1803 ##1%
1804 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1805 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1806 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1807 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1808 \def\bbl@encstring##1##2{%
1809 \bbl@foreach\bbl@sc@fontenc{%
1810 \bbl@ifunset{T@###1}%
1811 {}%
1812 {\ProvideTextCommand##1{###1}{##2}%
1813 \bbl@toglobal##1%
1814 \expandafter
1815 \bbl@toglobal\csname###1\string##1\endcsname}}}%
1816 \def\bbl@sctest{%
1817 \bbl@xin@{\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1818 \fi
1819 \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
1820 \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
1821 \let\AfterBabelCommands\bbl@aftercmds
1822 \let\SetString\bbl@setstring
1823 \let\bbl@stringdef\bbl@encstring
1824 \else % ie, strings=value
1825 \bbl@sctest
1826 \ifin@
1827 \let\AfterBabelCommands\bbl@aftercmds
1828 \let\SetString\bbl@setstring
1829 \let\bbl@stringdef\bbl@provstring
1830 \fi\fi\fi
1831 \bbl@scswitch
1832 \ifx\bbl@G\@empty
1833 \def\SetString##1##2{%
1834 \bbl@error{missing-group}{##1}{}}%

```

```

1835 \fi
1836 \ifx\@empty#1%
1837 \bbl@usehooks{defaultcommands}{}%
1838 \else
1839 \@expandtwoargs
1840 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1841 \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).

```

1842 \def\bbl@forlang#1#2{%
1843 \bbl@for#1\bbl@L{%
1844 \bbl@xin@{,#1,}{,\BabelLanguages,}%
1845 \ifin@#2\relax\fi}}
1846 \def\bbl@scswitch{%
1847 \bbl@forlang\bbl@tempa{%
1848 \ifx\bbl@G\@empty\else
1849 \ifx\SetString@gobbletwo\else
1850 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1851 \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
1852 \ifin@\else
1853 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1854 \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1855 \fi
1856 \fi
1857 \fi}}
1858 \AtEndOfPackage{%
1859 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\#2}}}%
1860 \let\bbl@scswitch\relax}
1861 \@onlypreamble\EndBabelCommands
1862 \def\EndBabelCommands{%
1863 \bbl@usehooks{stopcommands}{}%
1864 \endgroup
1865 \endgroup
1866 \bbl@scafter}
1867 \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.

```

1868 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1869 \bbl@forlang\bbl@tempa{%
1870 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1871 \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1872 {\bbl@exp{%
1873 \global\\bbl@add\<\bbl@G\bbl@tempa>{\bbl@scset\\#1\<\bbl@LC>}}}%
1874 }%
1875 \def\BabelString{#2}%
1876 \bbl@usehooks{stringprocess}{}%
1877 \expandafter\bbl@stringdef
1878 \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`.

```
1879 \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.

```
1880 << *Macros local to BabelCommands >> ≡
1881 \def\SetStringLoop##1##2{%
1882   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1883   \count@\z@
1884   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1885     \advance\count@\@ne
1886     \toks@\expandafter{\bbl@tempa}%
1887     \bbl@exp{%
1888       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1889       \count@=\the\count@\relax}}}%
1890 <</Macros local to BabelCommands >>
```

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

```
1891 \def\bbl@aftercmds#1{%
1892   \toks@\expandafter{\bbl@scafter#1}%
1893   \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.

```
1894 << *Macros local to BabelCommands >> ≡
1895 \newcommand\SetCase[3][{}]{%
1896   \def\bbl@tempa####1####2{%
1897     \ifx####1@empty\else
1898       \bbl@carg\bbl@add{extras\CurrentOption}{%
1899         \bbl@carg\babel@save{c__text_uppercase\_string####1_tl}%
1900         \bbl@carg\def{c__text_uppercase\_string####1_tl}{####2}%
1901         \bbl@carg\babel@save{c__text_lowercase\_string####2_tl}%
1902         \bbl@carg\def{c__text_lowercase\_string####2_tl}{####1}}%
1903       \expandafter\bbl@tempa
1904     \fi}%
1905   \bbl@tempa####1@empty@empty
1906   \bbl@carg\bbl@tglobal{extras\CurrentOption}}%
1907 <</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.

```
1908 << *Macros local to BabelCommands >> ≡
1909 \newcommand\SetHyphenMap[1]{%
1910   \bbl@forlang\bbl@tempa{%
1911     \expandafter\bbl@stringdef
1912     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1913 <</Macros local to BabelCommands >>
```

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

```
1914 \newcommand\BabelLower[2]{% one to one.
1915   \ifnum\lccode#1=#2\else
1916     \babel@savevariable{\lccode#1}%
1917     \lccode#1=#2\relax
1918   \fi}
1919 \newcommand\BabelLowerMM[4]{% many-to-many
1920   \@tempcnta=#1\relax
1921   \@tempcntb=#4\relax
1922   \def\bbl@tempa{%
1923     \ifnum\@tempcnta>#2\else
1924       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1925       \advance\@tempcnta#3\relax
```



```

1926      \advance\@tempcntb#3\relax
1927      \expandafter\bb\@tempa
1928      \fi}%
1929  \bb\@tempa}
1930 \newcommand\BabelLowerM0[4]{% many-to-one
1931  \@tempcnta=#1\relax
1932  \def\bb\@tempa{%
1933    \ifnum\@tempcnta>#2\else
1934      \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1935      \advance\@tempcnta#3
1936      \expandafter\bb\@tempa
1937      \fi}%
1938  \bb\@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1939 <<{*More package options}>> ≡
1940 \DeclareOption{hyphenmap=off}{\chardef\bb\@opt@hyphenmap\z@}
1941 \DeclareOption{hyphenmap=first}{\chardef\bb\@opt@hyphenmap\@ne}
1942 \DeclareOption{hyphenmap=select}{\chardef\bb\@opt@hyphenmap\tw@}
1943 \DeclareOption{hyphenmap=other}{\chardef\bb\@opt@hyphenmap\thr@}
1944 \DeclareOption{hyphenmap=other*}{\chardef\bb\@opt@hyphenmap4\relax}
1945 <</More package options>>

```

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

```

1946 \AtEndOfPackage{%
1947  \ifx\bb\@opt@hyphenmap\@undefined
1948    \bb\@xin@{,}{\bb\@language@opts}%
1949    \chardef\bb\@opt@hyphenmap\ifin@4\else\@ne\fi
1950  \fi}

```

## 4.11 Tailor captions

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.

```

1951 \newcommand\setlocalecaption{%^^A Catch typos.
1952  \@ifstar\bb\@setcaption@s\bb\@setcaption@x}
1953 \def\bb\@setcaption@x#1#2#3{% language caption-name string
1954  \bb\@trim@def\bb\@tempa{#2}%
1955  \bb\@xin@{.template}{\bb\@tempa}%
1956  \ifin@
1957    \bb\@ini@captions@template{#3}{#1}%
1958  \else
1959    \edef\bb\@tempd{%
1960      \expandafter\expandafter\expandafter
1961      \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1962    \bb\@xin@
1963      {\expandafter\string\csname #2name\endcsname}%
1964      {\bb\@tempd}%
1965    \ifin@ % Renew caption
1966      \bb\@xin@{\string\bb\@scset}{\bb\@tempd}%
1967      \ifin@
1968        \bb\@exp{%
1969          \\bb\@ifsamestring{\bb\@tempa}{\language}%
1970          {\bb\@scset\<#2name>\<#1#2name>}}%
1971        {}}%
1972      \else % Old way converts to new way
1973        \bb\@ifunset{#1#2name}%
1974        {\bb\@exp{%
1975          \\bb\@add\<captions#1>\def\<#2name>\<#1#2name>}}%
1976          \\bb\@ifsamestring{\bb\@tempa}{\language}%
1977          {\def\<#2name>\<#1#2name>}}%
1978          {}}}%
1979      {}%

```

```

1980     \fi
1981   \else
1982     \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
1983     \ifin@ % New way
1984       \bbl@exp{%
1985         \\bbl@add<captions#1>{\bbl@scset<#2name>\<#1#2name>}}%
1986         \\bbl@ifsamestring{\bbl@tempa}{\language}%
1987         {\bbl@scset<#2name>\<#1#2name>}}%
1988         {}}%
1989     \else % Old way, but defined in the new way
1990       \bbl@exp{%
1991         \\bbl@add<captions#1>{\def<#2name>\<#1#2name>}}%
1992         \\bbl@ifsamestring{\bbl@tempa}{\language}%
1993         {\def<#2name>\<#1#2name>}}%
1994         {}}%
1995     \fi%
1996   \fi
1997   \@namedef{#1#2name}{#3}%
1998   \toks@{\expandafter\bbl@captionslist}%
1999   \bbl@exp{\in@{\<#2name>}{\the\toks@}}%
2000   \ifin@\else
2001     \bbl@exp{\bbl@add\bbl@captionslist{\<#2name>}}%
2002     \bbl@toggle\bbl@captionslist
2003   \fi
2004 \fi}
2005 %^A \def\bbl@setcaption@#1#2#3{} % Not yet implemented (w/o 'name')

```

## 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`.

**\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.

```

2006 \bbl@trace{Macros related to glyphs}
2007 \def\set@low@box#1{\setbox\tw@hbox{,}\setbox\z@hbox{#1}%
2008   \dimen\z@ht\z@ \advance\dimen\z@ -\ht\tw@%
2009   \setbox\z@hbox{\lower\dimen\z@ \box\z@}\ht\z@ht\tw@ \dp\z@dp\tw@}

```

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

```

2010 \def\save@sf@q#1{\leavevmode
2011   \begingroup
2012   \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2013   \endgroup}

```

### 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.

```

2014 \ProvideTextCommand{\quotedblbase}{OT1}{%
2015   \save@sf@q{\set@low@box{\textquotedblright/}}%
2016   \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.

```

2017 \ProvideTextCommandDefault{\quotedblbase}{%
2018   \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```
2019 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2020   \save@sf@q{\set@low@box{\textquoteright\}%
2021     \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.

```
2022 \ProvideTextCommandDefault{\quotesinglbase}{%
2023   \UseTextSymbol{OT1}{\quotesinglbase}}
```

## **\guillemetleft**

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

```
2024 \ProvideTextCommand{\guillemetleft}{OT1}{%
2025   \ifmmode
2026     \ll
2027   \else
2028     \save@sf@q{\nobreak
2029       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2030   \fi}
2031 \ProvideTextCommand{\guillemetright}{OT1}{%
2032   \ifmmode
2033     \gg
2034   \else
2035     \save@sf@q{\nobreak
2036       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2037   \fi}
2038 \ProvideTextCommand{\guillemotleft}{OT1}{%
2039   \ifmmode
2040     \ll
2041   \else
2042     \save@sf@q{\nobreak
2043       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2044   \fi}
2045 \ProvideTextCommand{\guillemotright}{OT1}{%
2046   \ifmmode
2047     \gg
2048   \else
2049     \save@sf@q{\nobreak
2050       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2051   \fi}
```

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

```
2052 \ProvideTextCommandDefault{\guillemetleft}{%
2053   \UseTextSymbol{OT1}{\guillemetleft}}
2054 \ProvideTextCommandDefault{\guillemetright}{%
2055   \UseTextSymbol{OT1}{\guillemetright}}
2056 \ProvideTextCommandDefault{\guillemotleft}{%
2057   \UseTextSymbol{OT1}{\guillemotleft}}
2058 \ProvideTextCommandDefault{\guillemotright}{%
2059   \UseTextSymbol{OT1}{\guillemotright}}
```

## **\guilsinglleft**

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

```
2060 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2061   \ifmmode
2062     <%
2063   \else
2064     \save@sf@q{\nobreak
2065       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2066   \fi}
```

```

2066 \fi}
2067 \ProvideTextCommand{\guilsinglright}{OT1}{%
2068 \ifmmode
2069 >%
2070 \else
2071 \save@sf@q{\nobreak
2072 \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2073 \fi}

```

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

```

2074 \ProvideTextCommandDefault{\guilsinglleft}{%
2075 \UseTextSymbol{OT1}{\guilsinglleft}}
2076 \ProvideTextCommandDefault{\guilsinglright}{%
2077 \UseTextSymbol{OT1}{\guilsinglright}}

```

#### 4.12.2 Letters

##### **\ij**

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

```

2078 \DeclareTextCommand{\ij}{OT1}{%
2079 i\kern-0.02em\bbl@allowhyphens j}
2080 \DeclareTextCommand{\IJ}{OT1}{%
2081 I\kern-0.02em\bbl@allowhyphens J}
2082 \DeclareTextCommand{\ij}{T1}{\char188}
2083 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2084 \ProvideTextCommandDefault{\ij}{%
2085 \UseTextSymbol{OT1}{\ij}}
2086 \ProvideTextCommandDefault{\IJ}{%
2087 \UseTextSymbol{OT1}{\IJ}}

```

##### **\dj**

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

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

```

2088 \def\crrtic@{\hrule height0.1ex width0.3em}
2089 \def\crttic@{\hrule height0.1ex width0.33em}
2090 \def\ddj@{%
2091 \setbox0\hbox{d}\dimen@=\ht0
2092 \advance\dimen@lex
2093 \dimen@.45\dimen@
2094 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2095 \advance\dimen@ii.5ex
2096 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2097 \def\DDJ@{%
2098 \setbox0\hbox{D}\dimen@=.55\ht0
2099 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2100 \advance\dimen@ii.15ex % correction for the dash position
2101 \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2102 \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2103 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2104 %
2105 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2106 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```



**\frqq** The ‘french’ double guillemets.

```
2139 \ProvideTextCommandDefault{\flqq}{%
2140   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2141 \ProvideTextCommandDefault{\frqq}{%
2142   \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**

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

```
2143 \def\umlauthigh{%
2144   \def\bbl@umlauta##1{\leavevmode\bgroup%
2145     \accent\csname\fontencoding dqpos\endcsname
2146     ##1\bbl@allowhyphens\egroup}%
2147   \let\bbl@umlaute\bbl@umlauta}
2148 \def\umlautlow{%
2149   \def\bbl@umlauta{\protect\lower@umlaut}}
2150 \def\umlautel@low{%
2151   \def\bbl@umlaute{\protect\lower@umlaut}}
2152 \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.

```
2153 \expandafter\ifx\csname U@D\endcsname\relax
2154   \csname newdimen\endcsname U@D
2155 \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.

```
2156 \def\lower@umlaut#1{%
2157   \leavevmode\bgroup
2158   \U@D lex%
2159   {\setbox\z@\hbox{%
2160     \char\csname\fontencoding dqpos\endcsname}%
2161     \dimen@ -.45ex\advance\dimen@\ht\z@
2162     \ifdim lex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2163   \accent\csname\fontencoding dqpos\endcsname
2164   \fontdimen5\font\U@D #1%
2165   \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).

```
2166 \AtBeginDocument{%
2167   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2168   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
```

```

2169 \DeclareTextCompositeCommand{"}{OT1}{i}{\bbl@umlaut{i}}%
2170 \DeclareTextCompositeCommand{"}{OT1}{\i}{\bbl@umlaut{i}}%
2171 \DeclareTextCompositeCommand{"}{OT1}{o}{\bbl@umlaut{o}}%
2172 \DeclareTextCompositeCommand{"}{OT1}{u}{\bbl@umlaut{u}}%
2173 \DeclareTextCompositeCommand{"}{OT1}{A}{\bbl@umlaut{A}}%
2174 \DeclareTextCompositeCommand{"}{OT1}{E}{\bbl@umlaut{E}}%
2175 \DeclareTextCompositeCommand{"}{OT1}{I}{\bbl@umlaut{I}}%
2176 \DeclareTextCompositeCommand{"}{OT1}{O}{\bbl@umlaut{O}}%
2177 \DeclareTextCompositeCommand{"}{OT1}{U}{\bbl@umlaut{U}}

```

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

```

2178 \ifx\l@english\undefined
2179 \chardef\l@english\z@
2180 \fi
2181 % The following is used to cancel rules in ini files (see Amharic).
2182 \ifx\l@unhyphenated\undefined
2183 \newlanguage\l@unhyphenated
2184 \fi

```

### 4.13 Layout

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

```

2185 \bbl@trace{Bidi layout}
2186 \providecommand\IfBabelLayout[3]{#3}%
2187 </package | core>
2188 <*package>
2189 \newcommand\BabelPatchSection[1]{%
2190 \ifundefined{#1}{}%
2191 \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2192 \@namedef{#1}{%
2193 \ifstar{\bbl@presec@{#1}}%
2194 {\@dblarg{\bbl@presec@x{#1}}}}%
2195 \def\bbl@presec@x#1[#2]#3{%
2196 \bbl@exp{%
2197 \\\select@language@x{\bbl@main@language}%
2198 \\\bbl@cs{sspre@#1}%
2199 \\\bbl@cs{ss@#1}%
2200 [\\foreignlanguage{\language}\unexpanded{#2}}%
2201 {\\foreignlanguage{\language}\unexpanded{#3}}%
2202 \\\select@language@x{\language}}%
2203 \def\bbl@presec@#1#2{%
2204 \bbl@exp{%
2205 \\\select@language@x{\bbl@main@language}%
2206 \\\bbl@cs{sspre@#1}%
2207 \\\bbl@cs{ss@#1}%
2208 {\\foreignlanguage{\language}\unexpanded{#2}}%
2209 \\\select@language@x{\language}}%
2210 \IfBabelLayout{sectioning}%
2211 {\BabelPatchSection{part}%
2212 \BabelPatchSection{chapter}%
2213 \BabelPatchSection{section}%
2214 \BabelPatchSection{subsection}%
2215 \BabelPatchSection{subsubsection}%
2216 \BabelPatchSection{paragraph}%
2217 \BabelPatchSection{subparagraph}%
2218 \def\babel@toc#1{%
2219 \select@language@x{\bbl@main@language}}}%
2220 \IfBabelLayout{captions}%
2221 {\BabelPatchSection{caption}}}%
2222 </package>
2223 <*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.

```
2224 \bbl@trace{Input engine specific macros}
2225 \ifcase\bbl@engine
2226   \input txtbabel.def
2227 \or
2228   \input luababel.def
2229 \or
2230   \input xebabel.def
2231 \fi
2232 \providecommand\babelfont{\bbl@error{only-lua-xe}{}}{}{}
2233 \providecommand\babelprehyphenation{\bbl@error{only-lua}{}}{}{}
2234 \ifx\babelposthyphenation\undefined
2235   \let\babelposthyphenation\babelprehyphenation
2236   \let\babelpatterns\babelprehyphenation
2237   \let\babelcharproperty\babelprehyphenation
2238 \fi
2239 </package | core>
```

## 4.15 Creating and modifying languages

Continue with  $\TeX$  only.

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

```
2240 <*package>
2241 \bbl@trace{Creating languages and reading ini files}
2242 \let\bbl@extend@ini@gobble
2243 \newcommand\babelprovide[2][]{%
2244   \let\bbl@savelangname\language
2245   \edef\bbl@savelocaleid{\the\localeid}%
2246   % Set name and locale id
2247   \edef\language{#2}%
2248   \bbl@id@assign
2249   % Initialize keys
2250   \bbl@vforeach{captions,date,import,main,script,language,%
2251     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2252     mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2253     Alph,labels,labels*,calendar,date,casing,interchar}%
2254     {\bbl@csarg\let{KVP@##1}\@nnil}%
2255   \global\let\bbl@release@transforms\@empty
2256   \global\let\bbl@release@casing\@empty
2257   \let\bbl@calendars\@empty
2258   \global\let\bbl@inidata\@empty
2259   \global\let\bbl@extend@ini@gobble
2260   \global\let\bbl@included@inis\@empty
2261   \gdef\bbl@key@list{;}%
2262   \bbl@forkv{#1}{%
2263     \in@{/}{##1}% With /, (re)sets a value in the ini
2264     \ifin@
2265       \global\let\bbl@extend@ini\bbl@extend@ini@aux
2266       \bbl@renewinikey##1\@@{##2}%
2267     \else
2268       \bbl@csarg\ifx{KVP@##1}\@nnil\else
2269         \bbl@error{unknown-provide-key}{##1}{}{}%
2270       \fi
2271       \bbl@csarg\def{KVP@##1}{##2}%
2272     \fi}%
2273   \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2274   \bbl@ifunset{date#2}\z@{\bbl@ifunset{\bbl@llevel@#2}\@ne\tw@}%
2275   % == init ==
```



```

2276 \ifx\bb@screset\@undefined
2277 \bb@ldfinit
2278 \fi
2279 % == date (as option) ==
2280 % \ifx\bb@KVP@date\@nnil\else
2281 % \fi
2282 % ==
2283 \let\bb@lbkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2284 \ifcase\bb@howloaded
2285 \let\bb@lbkflag\@empty % new
2286 \else
2287 \ifx\bb@KVP@hyphenrules\@nnil\else
2288 \let\bb@lbkflag\@empty
2289 \fi
2290 \ifx\bb@KVP@import\@nnil\else
2291 \let\bb@lbkflag\@empty
2292 \fi
2293 \fi
2294 % == import, captions ==
2295 \ifx\bb@KVP@import\@nnil\else
2296 \bb@exp{\bb@ifblank{\bb@KVP@import}}%
2297 {\ifx\bb@initload\relax
2298 \begingroup
2299 \def\BabelBeforeIni##1##2{\gdef\bb@KVP@import{##1}\endinput}%
2300 \bb@input@texini{##2}%
2301 \endgroup
2302 \else
2303 \xdef\bb@KVP@import{\bb@initload}%
2304 \fi}%
2305 {}%
2306 \let\bb@KVP@date\@empty
2307 \fi
2308 \let\bb@KVP@captions@\bb@KVP@captions %^^A A dirty hack
2309 \ifx\bb@KVP@captions\@nnil
2310 \let\bb@KVP@captions\bb@KVP@import
2311 \fi
2312 % ==
2313 \ifx\bb@KVP@transforms\@nnil\else
2314 \bb@replace\bb@KVP@transforms{ },}%
2315 \fi
2316 % == Load ini ==
2317 \ifcase\bb@howloaded
2318 \bb@provide@new{##2}%
2319 \else
2320 \bb@ifblank{##1}%
2321 {}% With \bb@load@basic below
2322 {\bb@provide@renew{##2}}%
2323 \fi
2324 % == include == TODO
2325 % \ifx\bb@included@inis\@empty\else
2326 % \bb@replace\bb@included@inis{ },}%
2327 % \bb@foreach\bb@included@inis%
2328 % \openin\bb@readstream=babel-##1.ini
2329 % \bb@extend@ini{##2}%
2330 % \closein\bb@readstream
2331 % \fi
2332 % Post tasks
2333 % -----
2334 % == subsequent calls after the first provide for a locale ==
2335 \ifx\bb@inidata\@empty\else
2336 \bb@extend@ini{##2}%
2337 \fi
2338 % == ensure captions ==

```

```

2339 \ifx\bbk@KVP@captions\@nnil\else
2340 \bbk@ifunset{bbk@extracaps@#2}%
2341   {\bbk@exp{\bbk@babelensure[exclude=\\today]{#2}}}%
2342   {\bbk@exp{\bbk@babelensure[exclude=\\today,
2343     include=\bbk@extracaps@#2]}{#2}}%
2344 \bbk@ifunset{bbk@ensure@\language}%
2345   {\bbk@exp%
2346     \\\DeclareRobustCommand\<bbk@ensure@\language>[1]{%
2347       \\\foreignlanguage{\language}%
2348       {####1}}}%
2349   {}%
2350 \bbk@exp%
2351   \\\bbk@tglobal\<bbk@ensure@\language>%
2352   \\\bbk@tglobal\<bbk@ensure@\language\space>%
2353 \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.

```

2354 \bbk@load@basic{#2}%
2355 % == script, language ==
2356 % Override the values from ini or defines them
2357 \ifx\bbk@KVP@script\@nnil\else
2358   \bbk@csarg\edef{sname@#2}{\bbk@KVP@script}%
2359 \fi
2360 \ifx\bbk@KVP@language\@nnil\else
2361   \bbk@csarg\edef{lname@#2}{\bbk@KVP@language}%
2362 \fi
2363 \ifcase\bbk@engine\or
2364   \bbk@ifunset{bbk@chrng@\language}{}%
2365   {\directlua{
2366     Babel.set_chrngs_b('\bbk@cl{sbc}', '\bbk@cl{chrng}') }}%
2367 \fi
2368 % == onchar ==
2369 \ifx\bbk@KVP@onchar\@nnil\else
2370   \bbk@lua@hyphenate
2371   \bbk@exp%
2372     \\\AddToHook{env/document/before}{\select@language{#2}}}%
2373   \directlua{
2374     if Babel.locale_mapped == nil then
2375       Babel.locale_mapped = true
2376       Babel.linebreaking.add_before(Babel.locale_map, 1)
2377       Babel.loc_to_scr = {}
2378       Babel.chr_to_loc = Babel.chr_to_loc or {}
2379     end
2380     Babel.locale_props[\the\localeid].letters = false
2381   }%
2382   \bbk@xin@{ letters }{ \bbk@KVP@onchar\space}%
2383   \ifin@
2384     \directlua{
2385       Babel.locale_props[\the\localeid].letters = true
2386     }%
2387   \fi
2388   \bbk@xin@{ ids }{ \bbk@KVP@onchar\space}%
2389   \ifin@
2390     \ifx\bbk@starthyphens\@undefined % Needed if no explicit selection
2391       \AddBabelHook{babel-onchar}{beforestart}{\bbk@starthyphens}%
2392     \fi
2393     \bbk@exp{\bbk@add\bbk@starthyphens
2394       {\bbk@patterns@lua{\language}}}%
2395     %^A add error/warning if no script
2396     \directlua{
2397       if Babel.script_blocks['\bbk@cl{sbc}'] then

```

```

2398         Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbl@cl{sbc}']
2399         Babel.locale_props[\the\localeid].lg = \the\@nameuse{\@language}\space
2400     end
2401 }%
2402 \fi
2403 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2404 \ifin@
2405     \bbl@ifunset{\bbl@lsys\language}\bbl@provide@lsys{\language}}{}%
2406     \bbl@ifunset{\bbl@wdir\language}\bbl@provide@dirs{\language}}{}%
2407     \directlua{
2408         if Babel.script_blocks['\bbl@cl{sbc}'] then
2409             Babel.loc_to_scr[\the\localeid] =
2410                 Babel.script_blocks['\bbl@cl{sbc}']
2411         end}%
2412     \ifx\bbl@mapselect\undefined % TODO. almost the same as mapfont
2413     \AtBeginDocument{%
2414         \bbl@patchfont{\bbl@mapselect}}%
2415         {\selectfont}}%
2416     \def\bbl@mapselect{%
2417         \let\bbl@mapselect\relax
2418         \edef\bbl@prefontid{\fontid\font}}%
2419     \def\bbl@mapdir##1{%
2420         \begingroup
2421             \setbox\z@\hbox{% Force text mode
2422                 \def\language{##1}%
2423                 \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2424                 \bbl@switchfont
2425                 \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2426                     \directlua{
2427                         Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
2428                             [/\bbl@prefontid'] = \fontid\font\space}%
2429                     \fi}%
2430             \endgroup}%
2431     \fi
2432     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
2433 \fi
2434 % TODO - catch non-valid values
2435 \fi
2436 % == mapfont ==
2437 % For bidi texts, to switch the font based on direction
2438 \ifx\bbl@KVP@mapfont\@nnil\else
2439     \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}{}%
2440     {\bbl@error{unknown-mapfont}}{}{}%
2441     \bbl@ifunset{\bbl@lsys\language}\bbl@provide@lsys{\language}}{}%
2442     \bbl@ifunset{\bbl@wdir\language}\bbl@provide@dirs{\language}}{}%
2443 \ifx\bbl@mapselect\undefined % TODO. See onchar.
2444     \AtBeginDocument{%
2445         \bbl@patchfont{\bbl@mapselect}}%
2446         {\selectfont}}%
2447     \def\bbl@mapselect{%
2448         \let\bbl@mapselect\relax
2449         \edef\bbl@prefontid{\fontid\font}}%
2450     \def\bbl@mapdir##1{%
2451         {\def\language{##1}%
2452         \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2453         \bbl@switchfont
2454         \directlua{Babel.fontmap
2455             [\the\csname bbl@wdir@##1\endcsname]%
2456             [\bbl@prefontid]=\fontid\font}}}%
2457     \fi
2458     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
2459 \fi
2460 % == Line breaking: intraspace, intrapenalty ==

```

```

2461 % For CJK, East Asian, Southeast Asian, if interspace in ini
2462 \ifx\bbbl@KVP@intraspace\@nnil\else % We can override the ini or set
2463 \bbbl@csarg\edef{intsp@#2}{\bbbl@KVP@intraspace}%
2464 \fi
2465 \bbbl@provide@intraspace
2466 % == Line breaking: CJK quotes == %^^A -> @extras
2467 \ifcase\bbbl@engine\or
2468 \bbbl@xin@{/c}{\bbbl@cl{lnbrk}}%
2469 \ifin@
2470 \bbbl@ifunset{bbbl@quote@\language\name}{}%
2471 {\directlua{
2472 Babel.locale_props[\the\localeid].cjk_quotes = {}
2473 local cs = 'op'
2474 for c in string.utfvalues(%
2475 [[\csname bbbl@quote@\language\name\endcsname]]) do
2476 if Babel.cjk_characters[c].c == 'qu' then
2477 Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2478 end
2479 cs = ( cs == 'op') and 'cl' or 'op'
2480 end
2481 }}%
2482 \fi
2483 \fi
2484 % == Line breaking: justification ==
2485 \ifx\bbbl@KVP@justification\@nnil\else
2486 \let\bbbl@KVP@linebreaking\bbbl@KVP@justification
2487 \fi
2488 \ifx\bbbl@KVP@linebreaking\@nnil\else
2489 \bbbl@xin@{,\bbbl@KVP@linebreaking,}%
2490 {,elongated,kashida,cjk,padding,unhyphenated,}%
2491 \ifin@
2492 \bbbl@csarg\xdef
2493 {\lnbrk@\language\name}{\expandafter\@car\bbbl@KVP@linebreaking\@nil}%
2494 \fi
2495 \fi
2496 \bbbl@xin@{/e}{\bbbl@cl{lnbrk}}%
2497 \ifin@ \else \bbbl@xin@{/k}{\bbbl@cl{lnbrk}} \fi
2498 \ifin@ \bbbl@arabicjust \fi
2499 \bbbl@xin@{/p}{\bbbl@cl{lnbrk}}%
2500 \ifin@ \AtBeginDocument{\@nameuse{bbbl@tibetanjust}} \fi
2501 % == Line breaking: hyphenate.other.(locale|script) ==
2502 \ifx\bbbl@lbfkflag\@empty
2503 \bbbl@ifunset{bbbl@hyotl@\language\name}{}%
2504 {\bbbl@csarg\bbbl@replace{hyotl@\language\name}{ }{,}%
2505 \bbbl@startcommands*\language\name}{}%
2506 \bbbl@csarg\bbbl@foreach{hyotl@\language\name}{%
2507 \ifcase\bbbl@engine
2508 \ifnum##1<257
2509 \SetHyphenMap{\BabelLower{##1}{##1}}%
2510 \fi
2511 \else
2512 \SetHyphenMap{\BabelLower{##1}{##1}}%
2513 \fi}%
2514 \bbbl@endcommands}%
2515 \bbbl@ifunset{bbbl@hyots@\language\name}{}%
2516 {\bbbl@csarg\bbbl@replace{hyots@\language\name}{ }{,}%
2517 \bbbl@csarg\bbbl@foreach{hyots@\language\name}{%
2518 \ifcase\bbbl@engine
2519 \ifnum##1<257
2520 \global\lccode##1=##1\relax
2521 \fi
2522 \else
2523 \global\lccode##1=##1\relax

```

```

2524         \fi}}%
2525 \fi
2526 % == Counters: maparabic ==
2527 % Native digits, if provided in ini (TeX level, xe and lua)
2528 \ifcase\bbbl@engine\else
2529     \bbbl@ifunset{\bbbl@dgnat@\language\language}\fi%
2530     {\expandafter\ifx\csname \bbbl@dgnat@\language\language\endcsname\@empty\else
2531         \expandafter\expandafter\expandafter
2532         \bbbl@setdigits\csname \bbbl@dgnat@\language\language\endcsname
2533         \ifx\bbbl@KVP@maparabic\@nnil\else
2534             \ifx\bbbl@latinarabic\@undefined
2535                 \expandafter\let\expandafter\@arabic
2536                 \csname \bbbl@counter@\language\language\endcsname
2537             \else % ie, if layout=counters, which redefines \@arabic
2538                 \expandafter\let\expandafter\bbbl@latinarabic
2539                 \csname \bbbl@counter@\language\language\endcsname
2540             \fi
2541         \fi
2542     \fi}%
2543 \fi
2544 % == Counters: mapdigits ==
2545 % > luababel.def
2546 % == Counters: alph, Alph ==
2547 \ifx\bbbl@KVP@alph\@nnil\else
2548     \bbbl@exp{%
2549         \\bbbl@add\<\bbbl@preextras@\language\language>{%
2550             \\bbabel@save\\@alph
2551             \let\\@alph\<\bbbl@cntr@\bbbl@KVP@alph @\language\language>}}%
2552 \fi
2553 \ifx\bbbl@KVP@Alph\@nnil\else
2554     \bbbl@exp{%
2555         \\bbbl@add\<\bbbl@preextras@\language\language>{%
2556             \\bbabel@save\\@Alph
2557             \let\\@Alph\<\bbbl@cntr@\bbbl@KVP@Alph @\language\language>}}%
2558 \fi
2559 % == Casing ==
2560 \bbbl@release@casing
2561 \ifx\bbbl@KVP@casing\@nnil\else
2562     \bbbl@csarg\xdef{casing@\language\language}%
2563     {\@nameuse{\bbbl@casing@\language\language}\bbbl@maybextx\bbbl@KVP@casing}%
2564 \fi
2565 % == Calendars ==
2566 \ifx\bbbl@KVP@calendar\@nnil
2567     \edef\bbbl@KVP@calendar{\bbbl@cl{calpr}}%
2568 \fi
2569 \def\bbbl@tempe##1##2\@{##1}% Get first calendar
2570 \def\bbbl@tempa{##1}%
2571 \bbbl@exp{\bbbl@tempe\bbbl@KVP@calendar\space\@}%
2572 \def\bbbl@tempe##1.##2.##3\@{##1}%
2573 \def\bbbl@tempc{##1}%
2574 \def\bbbl@tempb{##2}%
2575 \expandafter\bbbl@tempe\bbbl@tempa.\@
2576 \bbbl@csarg\xdef{calpr@\language\language}%
2577 \ifx\bbbl@tempc\@empty\else
2578     calendar=\bbbl@tempc
2579 \fi
2580 \ifx\bbbl@tempb\@empty\else
2581     ,variant=\bbbl@tempb
2582 \fi}%
2583 % == engine specific extensions ==
2584 % Defined in XXXbabel.def
2585 \bbbl@provide@extra{#2}%
2586 % == require.babel in ini ==

```

```

2587 % To load or reload the babel-*.tex, if require.babel in ini
2588 \ifx\babel@beforestart\relax\else % But not in doc aux or body
2589   \babel@ifunset{babel@rqtex@\language}\relax\else
2590     {\expandafter\ifx\csname babel@rqtex@\language\endcsname\@empty\else
2591       \let\BabelBeforeIni@gobbletwo
2592       \chardef\atcatcode=\catcode\@
2593       \catcode\@=11\relax
2594       \def\CurrentOption{#2}%
2595       \babel@input@texini{\babel@cs{rqtex@\language}}%
2596       \catcode\@=\atcatcode
2597       \let\atcatcode\relax
2598       \global\babel@csarg\let{rqtex@\language}\relax
2599     \fi}%
2600 \babel@foreach\babel@calendars{%
2601   \babel@ifunset{babel@ca##1}{%
2602     \chardef\atcatcode=\catcode\@
2603     \catcode\@=11\relax
2604     \InputIfFileExists{babel-ca-##1.tex}{\relax}%
2605     \catcode\@=\atcatcode
2606     \let\atcatcode\relax}%
2607   {}}%
2608 \fi
2609 % == frenchspacing ==
2610 \ifcase\babel@howloaded\in@true\else\in@false\fi
2611 \ifin@else\babel@xin@{typography/frenchspacing}{\babel@key@list}\fi
2612 \ifin@
2613   \babel@extras@wrap{\babel@pre@fs}%
2614   {\babel@pre@fs}%
2615   {\babel@post@fs}%
2616 \fi
2617 % == transforms ==
2618 % > luababel.def
2619 \def\CurrentOption{#2}%
2620 \@nameuse{babel@icsave@#2}%
2621 % == main ==
2622 \ifx\babel@KVP@main\@nnil % Restore only if not 'main'
2623   \let\language\babel@savelangname
2624   \chardef\localeid\babel@savelocaleid\relax
2625 \fi
2626 % == hyphenrules (apply if current) ==
2627 \ifx\babel@KVP@hyphenrules\@nnil\else
2628   \ifnum\babel@savelocaleid=\localeid
2629     \language\@nameuse{l@\language}%
2630   \fi
2631 \fi}

```

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

```

2632 \def\babel@provide@new#1{%
2633   \@namedef{date#1}{\relax}% marks lang exists - required by \StartBabelCommands
2634   \@namedef{extras#1}{\relax}%
2635   \@namedef{noextras#1}{\relax}%
2636   \babel@startcommands*{#1}{captions}%
2637   \ifx\babel@KVP@captions\@nnil % and also if import, implicit
2638     \def\babel@tempb##1{% elt for \babel@captionslist
2639       \ifx##1\@nnil\else
2640         \babel@exp{%
2641           \\SetString\\##1%
2642           \\babel@nocaption{\babel@stripslash##1}{#1\babel@stripslash##1}}%
2643         \expandafter\babel@tempb
2644       \fi}%
2645   \expandafter\babel@tempb\babel@captionslist\@nnil
2646   \else

```

```

2647 \ifx\bbbl@initoload\relax
2648 \bbbl@read@ini{\bbbl@KVP@captions}2% % Here letters cat = 11
2649 \else
2650 \bbbl@read@ini{\bbbl@initoload}2% % Same
2651 \fi
2652 \fi
2653 \StartBabelCommands*{#1}{date}%
2654 \ifx\bbbl@KVP@date\@nnil
2655 \bbbl@exp{%
2656 \\\SetString\\today{\bbbl@nocaption{today}{#1today}}}%
2657 \else
2658 \bbbl@savetoday
2659 \bbbl@savestate
2660 \fi
2661 \bbbl@endcommands
2662 \bbbl@load@basic{#1}%
2663 % == hyphenmins == (only if new)
2664 \bbbl@exp{%
2665 \gdef\<#1hyphenmins>{%
2666 {\bbbl@ifunset{\bbbl@lfthm{#1}{2}}{\bbbl@cs{lfthm{#1}}}%
2667 {\bbbl@ifunset{\bbbl@rgthm{#1}{3}}{\bbbl@cs{rgthm{#1}}}}}%
2668 % == hyphenrules (also in renew) ==
2669 \bbbl@provide@hyphens{#1}%
2670 \ifx\bbbl@KVP@main\@nnil\else
2671 \expandafter\main@language\expandafter{#1}%
2672 \fi}
2673 %
2674 \def\bbbl@provide@renew#1{%
2675 \ifx\bbbl@KVP@captions\@nnil\else
2676 \StartBabelCommands*{#1}{captions}%
2677 \bbbl@read@ini{\bbbl@KVP@captions}2% % Here all letters cat = 11
2678 \EndBabelCommands
2679 \fi
2680 \ifx\bbbl@KVP@date\@nnil\else
2681 \StartBabelCommands*{#1}{date}%
2682 \bbbl@savetoday
2683 \bbbl@savestate
2684 \EndBabelCommands
2685 \fi
2686 % == hyphenrules (also in new) ==
2687 \ifx\bbbl@lbfkflag\@empty
2688 \bbbl@provide@hyphens{#1}%
2689 \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.

```

2690 \def\bbbl@load@basic#1{%
2691 \ifcase\bbbl@howloaded\or\or
2692 \ifcase\csname bbl@llevel@\language\endcsname
2693 \bbbl@csarg\let\lname@\language\relax
2694 \fi
2695 \fi
2696 \bbbl@ifunset{\bbbl@lname{#1}}%
2697 {\def\BabelBeforeIni##1##2{%
2698 \begingroup
2699 \let\bbbl@ini@captions@aux\@gobbletwo
2700 \def\bbbl@inidate ####1.####2.####3.####4\relax ####5####6}%
2701 \bbbl@read@ini{##1}1%
2702 \ifx\bbbl@initoload\relax\endinput\fi
2703 \endgroup}%
2704 \begingroup % boxed, to avoid extra spaces:
2705 \ifx\bbbl@initoload\relax

```

```

2706      \bbl@input@texini{#1}%
2707      \else
2708      \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
2709      \fi
2710      \endgroup}%
2711      {}%

```

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`.

```

2712 \def\bbl@provide@hyphens#1{%
2713   \@tempcnta\m@ne % a flag
2714   \ifx\bbl@KVP@hyphenrules\@nnil\else
2715     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2716     \bbl@foreach\bbl@KVP@hyphenrules{%
2717       \ifnum\@tempcnta=\m@ne % if not yet found
2718         \bbl@ifsamestring{##1}{+}%
2719         {\bbl@carg\addlanguage{l@##1}}%
2720         {}%
2721         \bbl@ifunset{l@##1}% After a possible +
2722         {}%
2723         {\@tempcnta\@nameuse{l@##1}}%
2724       \fi}%
2725   \ifnum\@tempcnta=\m@ne
2726     \bbl@warning{%
2727       Requested 'hyphenrules' for '\language' not found:\\%
2728       \bbl@KVP@hyphenrules.\\%
2729       Using the default value. Reported}%
2730     \fi
2731   \fi
2732   \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found
2733     \ifx\bbl@KVP@captions@\@nnil % TODO. Hackish. See above.
2734       \bbl@ifunset{\bbl@hyphr#1}{}% use value in ini, if exists
2735       {\bbl@exp{\bbl@ifblank{\bbl@cs{hyphr#1}}}%
2736       {}%
2737       {\bbl@ifunset{l@\bbl@cl{hyphr}}}%
2738       {}% if hyphenrules found:
2739       {\@tempcnta\@nameuse{l@\bbl@cl{hyphr}}}%
2740     \fi
2741   \fi
2742   \bbl@ifunset{l@#1}%
2743   {\ifnum\@tempcnta=\m@ne
2744     \bbl@carg\adddialect{l@#1}\language
2745     \else
2746     \bbl@carg\adddialect{l@#1}\@tempcnta
2747     \fi}%
2748   {\ifnum\@tempcnta=\m@ne\else
2749     \global\bbl@carg\chardef{l@#1}\@tempcnta
2750     \fi}}

```

The reader of `babel-...tex` files. We reset temporarily some catcodes (and make sure no space is accidentally inserted).

```

2751 \def\bbl@input@texini#1{%
2752   \bbl@bsphack
2753   \bbl@exp{%
2754     \catcode`\%%=14 \catcode`\%%=0
2755     \catcode`\%={1 \catcode`\%}=2
2756     \lowercase{\InputIfFileExists{babel-#1.tex}}}%
2757     \catcode`\%%=\the\catcode`\%\relax
2758     \catcode`\%=\the\catcode`\%\relax
2759     \catcode`\%={\the\catcode`\%\relax
2760     \catcode`\%=\the\catcode`\%\relax}%
2761   \bbl@esphack}

```

The following macros read and store ini files (but don't process them). For each line, there are 3



possible actions: ignore if starts with ;, switch section if starts with [, and store otherwise. There are used in the first step of \bbl@read@ini.

```

2762 \def\bbl@iniline#1\bbl@iniline{%
2763   \@ifnextchar[\bbl@inisect{\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
2764 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2765 \def\bbl@iniskip#1\@@{%      if starts with ;
2766 \def\bbl@inistore#1=#2\@@{%    full (default)
2767   \bbl@trim@def\bbl@tempa{#1}%
2768   \bbl@trim\toks@{#2}%
2769   \bbl@xin@{;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2770   \ifin@else
2771     \bbl@xin@{,identification/include.}%
2772     {,\bbl@section/\bbl@tempa}%
2773   \ifin@\xdef\bbl@included@inis{\the\toks@}\fi
2774   \bbl@exp{%
2775     \\g@addto@macro\\bbl@inidata{%
2776       \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2777   \fi}
2778 \def\bbl@inistore@min#1=#2\@@{% minimal (maybe set in \bbl@read@ini)
2779   \bbl@trim@def\bbl@tempa{#1}%
2780   \bbl@trim\toks@{#2}%
2781   \bbl@xin@{.identification.}{.\bbl@section.}%
2782   \ifin@
2783     \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2784       \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2785   \fi}

```

## 4.16 Main loop in ‘provide’

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.

```

2786 \def\bbl@loop@ini{%
2787   \loop
2788     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2789     \endlinechar\m@ne
2790     \read\bbl@readstream to \bbl@line
2791     \endlinechar\^^M
2792     \ifx\bbl@line\empty\else
2793       \expandafter\bbl@iniline\bbl@line\bbl@iniline
2794     \fi
2795   \repeat}
2796 \ifx\bbl@readstream\undefined
2797   \csname newread\endcsname\bbl@readstream
2798 \fi
2799 \def\bbl@read@ini#1#2{%
2800   \global\let\bbl@extend@ini@gobble
2801   \openin\bbl@readstream=babel-#1.ini
2802   \ifeof\bbl@readstream
2803     \bbl@error{no-ini-file}{#1}{}{}%
2804   \else
2805     % == Store ini data in \bbl@inidata ==
2806     \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2807     \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2808     \bbl@info{Importing
2809       \ifcase#2font and identification \or basic \fi
2810       data for \languagename\\%
2811       from babel-#1.ini. Reported}%
2812     \ifnum#2=\z@

```

```

2813 \global\let\bbl@inidata\@empty
2814 \let\bbl@inistore\bbl@inistore@min % Remember it's local
2815 \fi
2816 \def\bbl@section{identification}%
2817 \bbl@exp{\bbl@inistore tag.ini=#1\\@@}%
2818 \bbl@inistore load.level=#2\\@@
2819 \bbl@loop@ini
2820 % == Process stored data ==
2821 \bbl@csarg\xdef{lini@language}{#1}%
2822 \bbl@read@ini@aux
2823 % == 'Export' data ==
2824 \bbl@ini@exports{#2}%
2825 \global\bbl@csarg\let{inidata@language}\bbl@inidata
2826 \global\let\bbl@inidata\@empty
2827 \bbl@exp{\bbl@add@list\bbl@ini@loaded{language}}%
2828 \bbl@tglobal\bbl@ini@loaded
2829 \fi
2830 \closein\bbl@readstream}
2831 \def\bbl@read@ini@aux{%
2832 \let\bbl@savestrings\@empty
2833 \let\bbl@savetoday\@empty
2834 \let\bbl@savestate\@empty
2835 \def\bbl@elt##1##2##3{%
2836 \def\bbl@section{##1}%
2837 \in@{=date.}{=##1}% Find a better place
2838 \ifin@
2839 \bbl@ifunset{bbl@inikv@##1}%
2840 {\bbl@ini@calendar{##1}}%
2841 {}%
2842 \fi
2843 \bbl@ifunset{bbl@inikv@##1}{}%
2844 {\csname bbl@inikv@##1\endcsname{##2}{##3}}%
2845 \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.

```

2846 \def\bbl@extend@ini@aux#1{%
2847 \bbl@startcommands*{#1}{captions}%
2848 % Activate captions/... and modify exports
2849 \bbl@csarg\def{inikv@captions.licr}##1##2{%
2850 \setlocalecaption{#1}{##1}{##2}}%
2851 \def\bbl@inikv@captions##1##2{%
2852 \bbl@ini@captions@aux{##1}{##2}}%
2853 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2854 \def\bbl@exportkey##1##2##3{%
2855 \bbl@ifunset{bbl@kv@##2}{}%
2856 {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2857 \bbl@exp{\global\let<bbl@##1@language>\<bbl@kv@##2>}}%
2858 \fi}}%
2859 % As with \bbl@read@ini, but with some changes
2860 \bbl@read@ini@aux
2861 \bbl@ini@exports\tw@
2862 % Update inidata@lang by pretending the ini is read.
2863 \def\bbl@elt##1##2##3{%
2864 \def\bbl@section{##1}%
2865 \bbl@iniline##2=##3\bbl@iniline}%
2866 \csname bbl@inidata@#1\endcsname
2867 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2868 \StartBabelCommands*{#1}{date}% And from the import stuff
2869 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2870 \bbl@savetoday
2871 \bbl@savestate
2872 \bbl@endcommands}

```

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

```

2873 \def\bbl@ini@calendar#1{%
2874 \lowercase{\def\bbl@tempa{=#1=}}%
2875 \bbl@replace\bbl@tempa{=date.gregorian}{}%
2876 \bbl@replace\bbl@tempa{=date.}{}%
2877 \in@{.licr=}{#1=}%
2878 \ifin@
2879 \ifcase\bbl@engine
2880 \bbl@replace\bbl@tempa{.licr=}{}%
2881 \else
2882 \let\bbl@tempa\relax
2883 \fi
2884 \fi
2885 \ifx\bbl@tempa\relax\else
2886 \bbl@replace\bbl@tempa{=}{}%
2887 \ifx\bbl@tempa\empty\else
2888 \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2889 \fi
2890 \bbl@exp{%
2891 \def<bbl@inikv#1>###1###2{%
2892 \\bbl@inidate###1...\relax{###2}{\bbl@tempa}}}%
2893 \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).

```

2894 \def\bbl@renewinikey#1/#2\@@#3{%
2895 \edef\bbl@tempa{\zap@space #1 \@empty}% section
2896 \edef\bbl@tempb{\zap@space #2 \@empty}% key
2897 \bbl@trim\toks@{#3}% value
2898 \bbl@exp{%
2899 \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2900 \\g@addto@macro\\bbl@inidata{%
2901 \\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.

```

2902 \def\bbl@exportkey#1#2#3{%
2903 \bbl@ifunset{bbl@kv@#2}%
2904 {\bbl@csarg\gdef{#1@\language}\@empty}%
2905 {\expandafter\ifx\csname bbl@kv@#2\endcsname\empty
2906 \bbl@csarg\gdef{#1@\language}\@empty}%
2907 \else
2908 \bbl@exp{\global\let<bbl@#1@\language><bbl@kv@#2>}%
2909 \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.

```

2910 \def\bbl@iniwarning#1{%
2911 \bbl@ifunset{bbl@kv@identification.warning#1}{}%
2912 {\bbl@warning{%
2913 From babel-\bbl@cs{lini@\language}.ini:\\%
2914 \bbl@cs{kv@identification.warning#1}\\%
2915 Reported }}}
2916 %
2917 \let\bbl@release@transforms\empty
2918 \let\bbl@release@casing\empty
2919 \def\bbl@ini@exports#1{%

```

```

2920 % Identification always exported
2921 \bbl@iniwarning{}%
2922 \ifcase\bbl@engine
2923   \bbl@iniwarning{\pdflatex}%
2924 \or
2925   \bbl@iniwarning{\lua\latex}%
2926 \or
2927   \bbl@iniwarning{\xel\latex}%
2928 \fi%
2929 \bbl@exportkey{lllevel}{identification.load.level}{}%
2930 \bbl@exportkey{elname}{identification.name.english}{}%
2931 \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
2932   {\csname bbl@elname\language\endcsname}}%
2933 \bbl@exportkey{tbc}{identification.tag.bcp47}{}%
2934 % Somewhat hackish. TODO:
2935 \bbl@exportkey{casing}{identification.tag.bcp47}{}%
2936 \bbl@exportkey{lbc}{identification.language.tag.bcp47}{}%
2937 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2938 \bbl@exportkey{esname}{identification.script.name}{}%
2939 \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
2940   {\csname bbl@esname\language\endcsname}}%
2941 \bbl@exportkey{sbc}{identification.script.tag.bcp47}{}%
2942 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2943 \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2944 \bbl@exportkey{vbc}{identification.variant.tag.bcp47}{}%
2945 \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%
2946 \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
2947 \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
2948 % Also maps bcp47 -> language
2949 \ifbbl@bcptoname
2950   \bbl@csarg\xdef{bcp@map@bbl@cl{tbc}}{\language}%
2951 \fi
2952 \ifcase\bbl@engine\or
2953   \directlua{%
2954     Babel.locale_props[\the\bbl@cs{id@}\language]}.script
2955     = '\bbl@cl{sbc}}'%
2956 \fi
2957 % Conditional
2958 \ifnum#1>\z@ % 0 = only info, 1, 2 = basic, (re)new
2959   \bbl@exportkey{calpr}{date.calendar.preferred}{}%
2960   \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2961   \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2962   \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
2963   \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2964   \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2965   \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2966   \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2967   \bbl@exportkey{intsp}{typography.intraspaces}{}%
2968   \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2969   \bbl@exportkey{chrng}{characters.ranges}{}%
2970   \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2971   \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2972   \ifnum#1=\tw@ % only (re)new
2973     \bbl@exportkey{rtex}{identification.require.babel}{}%
2974     \bbl@tglobal\bbl@savetoday
2975     \bbl@tglobal\bbl@savestate
2976     \bbl@savestrings
2977   \fi
2978 \fi}

```

## 4.17 Processing keys in ini

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

```

2979 \def\bbl@inikv#1#2{%      key=value
2980 \toks@{#2}%              This hides #'s from ini values
2981 \bbl@csarg\edef{kv@\bbl@section.#1}{\the\toks@}}

```

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

```

2982 \let\bbl@inikv@identification\bbl@inikv
2983 \let\bbl@inikv@date\bbl@inikv
2984 \let\bbl@inikv@typography\bbl@inikv
2985 \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.

```

2986 \def\bbl@maybextx{-\bbl@csarg\ifx{extx@\languagename}\@empty x-\fi}
2987 \def\bbl@inikv@characters#1#2{%
2988 \bbl@ifsamestring{#1}{casing}% eg, casing = uV
2989 {\bbl@exp{%
2990 \\\g@addto@macro{\bbl@release@casing{%
2991 \\\bbl@casemapping}{\languagename}{\unexpanded{#2}}}}}%
2992 {\in@{casing.}{#1}% eg, casing.Uv = uV
2993 \ifin@
2994 \lowercase{\def\bbl@tempb{#1}}%
2995 \bbl@replace\bbl@tempb{casing.}{}%
2996 \bbl@exp{\\\g@addto@macro{\bbl@release@casing{%
2997 \\\bbl@casemapping
2998 {\\\bbl@maybextx\bbl@tempb}{\languagename}{\unexpanded{#2}}}}}%
2999 \else
3000 \bbl@inikv{#1}{#2}%
3001 \fi}}

```

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

```

3002 \def\bbl@inikv@counters#1#2{%
3003 \bbl@ifsamestring{#1}{digits}%
3004 {\bbl@error{digits-is-reserved}{}}}%
3005 {}%
3006 \def\bbl@tempc{#1}%
3007 \bbl@trim@def{\bbl@tempb*}{#2}%
3008 \in@{.1$}{#1$}%
3009 \ifin@
3010 \bbl@replace\bbl@tempc{.1}{}%
3011 \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\languagename}{%
3012 \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3013 \fi
3014 \in@{.F.}{#1}%
3015 \ifin@\else\in@{.S.}{#1}\fi
3016 \ifin@
3017 \bbl@csarg\protected@xdef{cntr@#1@\languagename}{\bbl@tempb*}%
3018 \else
3019 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3020 \expandafter\bbl@buildifcase\bbl@tempb* \\\ % Space after \\\
3021 \bbl@csarg{\global\expandafter\let}{cntr@#1@\languagename}\bbl@tempa
3022 \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.

```

3023 \ifcase\bbl@engine
3024 \bbl@csarg\def{inikv@captions.licr}#1#2{%
3025 \bbl@ini@captions@aux{#1}{#2}}
3026 \else
3027 \def\bbl@inikv@captions#1#2{%
3028 \bbl@ini@captions@aux{#1}{#2}}
3029 \fi

```

The auxiliary macro for captions define `\captionname`.

```

3030 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3031   \bbl@replace\bbl@tempa{.template}{}%
3032   \def\bbl@toreplace{#1}{}%
3033   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3034   \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3035   \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3036   \bbl@replace\bbl@toreplace{[ ]}{name\endcsname{}}%
3037   \bbl@replace\bbl@toreplace{[ ]}{\endcsname{}}%
3038   \bbl@xin@{,\bbl@tempa,},{,chapter,appendix,part,}%
3039   \ifin@
3040     \@nameuse{\bbl@patch\bbl@tempa}%
3041     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3042   \fi
3043   \bbl@xin@{,\bbl@tempa,},{,figure,table,}%
3044   \ifin@
3045     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3046     \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
3047       \\bbl@ifunset{\bbl@\bbl@tempa fmt@\language\language}%
3048       {[fnum@\bbl@tempa]}%
3049       {\\@nameuse{\bbl@\bbl@tempa fmt@\language\language}}}%
3050     \fi}
3051 \def\bbl@ini@captions@aux#1#2{%
3052   \bbl@trim\def\bbl@tempa{#1}%
3053   \bbl@xin@{.template}{\bbl@tempa}%
3054   \ifin@
3055     \bbl@ini@captions@template{#2}\language\language
3056   \else
3057     \bbl@ifblank{#2}%
3058     {\bbl@exp{%
3059       \toks@{\\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
3060     {\bbl@trim\toks@{#2}}%
3061     \bbl@exp{%
3062       \\bbl@add\\bbl@savestrings{%
3063         \\SetString\<\bbl@tempa name>{\the\toks@}}%
3064       \toks@\xexpandafter{\bbl@captionslist}%
3065       \bbl@exp{\\in@{\<\bbl@tempa name>}{\the\toks@}}%
3066       \ifin@\\else
3067         \bbl@exp{%
3068           \\bbl@add\<\bbl@extracaps@\language\language>{\<\bbl@tempa name>}%
3069           \\bbl@to\global\<\bbl@extracaps@\language\language>}%
3070         \fi
3071       \fi}

```

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

```

3072 \def\bbl@list@the{%
3073   part,chapter,section,subsection,subsubsection,paragraph,%
3074   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3075   table,page,footnote,mpfootnote,mpfn}
3076 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3077   \bbl@ifunset{\bbl@map@#1@\language\language}%
3078   {\@nameuse{#1}}%
3079   {\@nameuse{\bbl@map@#1@\language\language}}}
3080 \def\bbl@inikv@labels#1#2{%
3081   \in@{.map}{#1}%
3082   \ifin@
3083     \ifx\bbl@KVP@labels\@nnil\else
3084       \bbl@xin@{ map }{\bbl@KVP@labels\space}%
3085       \ifin@
3086         \def\bbl@tempc{#1}%
3087         \bbl@replace\bbl@tempc{.map}{}%
3088         \in@{,#2,},{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3089         \bbl@exp{%

```

```

3090 \gdef\<bbl@map@bbl@tempc @\language>%
3091 {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3092 \bbl@foreach\bbl@list@the{%
3093 \bbl@ifunset{the##1}{%
3094 {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3095 \bbl@exp{%
3096 \\bbl@sreplace\<the##1>%
3097 {\<bbl@tempc>{##1}}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3098 \\bbl@sreplace\<the##1>%
3099 {\<\@empty @\bbl@tempc>\<c@##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3100 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3101 \toks@ \expandafter\expandafter\expandafter{%
3102 \csname the##1\endcsname}%
3103 \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3104 \fi}}%
3105 \fi
3106 \fi
3107 %
3108 \else
3109 %
3110 % The following code is still under study. You can test it and make
3111 % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3112 % language dependent.
3113 \in@{enumerate.}{#1}%
3114 \ifin@
3115 \def\bbl@tempa{#1}%
3116 \bbl@replace\bbl@tempa{enumerate.}{}%
3117 \def\bbl@toreplace{#2}%
3118 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3119 \bbl@replace\bbl@toreplace{[]}{\csname the}%
3120 \bbl@replace\bbl@toreplace{[]}{\endcsname{}}%
3121 \toks@ \expandafter{\bbl@toreplace}%
3122 % TODO. Execute only once:
3123 \bbl@exp{%
3124 \\bbl@add\<extras\language>{%
3125 \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3126 \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3127 \\bbl@tglobal\<extras\language>}%
3128 \fi
3129 \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.

```

3130 \def\bbl@chapttype{chapter}
3131 \ifx\@makechapterhead\@undefined
3132 \let\bbl@patchchapter\relax
3133 \else\ifx\thechapter\@undefined
3134 \let\bbl@patchchapter\relax
3135 \else\ifx\ps@headings\@undefined
3136 \let\bbl@patchchapter\relax
3137 \else
3138 \def\bbl@patchchapter{%
3139 \global\let\bbl@patchchapter\relax
3140 \gdef\bbl@chfmt{%
3141 \bbl@ifunset{bbl@\bbl@chapttype fmt@\language}%
3142 {\@chapapp\space\thechapter}
3143 {\@nameuse{bbl@\bbl@chapttype fmt@\language}}}
3144 \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3145 \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3146 \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3147 \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%

```

```

3148 \bbl@tglobal\appendix
3149 \bbl@tglobal\ps@headings
3150 \bbl@tglobal\chaptermark
3151 \bbl@tglobal\@makechapterhead}
3152 \let\bbl@patchappendix\bbl@patchchapter
3153 \fi\fi\fi
3154 \ifx\@part\@undefined
3155 \let\bbl@patchpart\relax
3156 \else
3157 \def\bbl@patchpart{%
3158 \global\let\bbl@patchpart\relax
3159 \gdef\bbl@partformat{%
3160 \bbl@ifunset\bbl@partfmt@\language\name}%
3161 {\partname\nobreakspace\thepart}
3162 {\@nameuse\bbl@partfmt@\language\name}}
3163 \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3164 \bbl@tglobal\@part}
3165 \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

```

3166 \let\bbl@calendar\@empty
3167 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3168 \def\bbl@localedate#1#2#3#4{%
3169 \begingroup
3170 \edef\bbl@they{#2}%
3171 \edef\bbl@them{#3}%
3172 \edef\bbl@thed{#4}%
3173 \edef\bbl@tempe{%
3174 \bbl@ifunset\bbl@calpr@\language\name}{\bbl@c\calpr}},%
3175 #1}%
3176 \bbl@replace\bbl@tempe{ }{}%
3177 \bbl@replace\bbl@tempe{CONVERT}{convert=}% Hackish
3178 \bbl@replace\bbl@tempe{convert}{convert=}%
3179 \let\bbl@ld@calendar\@empty
3180 \let\bbl@ld@variant\@empty
3181 \let\bbl@ld@convert\relax
3182 \def\bbl@tempb##1=##2\@{\@namedef\bbl@ld@##1}{##2}}%
3183 \bbl@foreach\bbl@tempe{\bbl@tempb##1\@}%
3184 \bbl@replace\bbl@ld@calendar{gregorian}{}%
3185 \ifx\bbl@ld@calendar\@empty\else
3186 \ifx\bbl@ld@convert\relax\else
3187 \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3188 {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3189 \fi
3190 \fi
3191 \@nameuse\bbl@precalendar}% Remove, eg, +, -civil (-ca-islamic)
3192 \edef\bbl@calendar{% Used in \month..., too
3193 \bbl@ld@calendar
3194 \ifx\bbl@ld@variant\@empty\else
3195 .\bbl@ld@variant
3196 \fi}%
3197 \bbl@cased
3198 {\@nameuse\bbl@date@\language\name @\bbl@calendar}%
3199 \bbl@they\bbl@them\bbl@thed}%
3200 \endgroup}
3201 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3202 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3203 \bbl@trim\def\bbl@tempa{#1.#2}%
3204 \bbl@ifsamestring\bbl@tempa{months.wide}% to savedate
3205 {\bbl@trim\def\bbl@tempa{#3}%
3206 \bbl@trim\toks{#5}%
3207 \@temptokena\expandafter\bbl@savestate}%

```



```

3208 \bbl@exp{% Reverse order - in ini last wins
3209 \def\\bbl@savestate{%
3210 \\\SetString<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3211 \the@temptokena}}}%
3212 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3213 {\lowercase{\def\bbl@tempb{#6}}}%
3214 \bbl@trim@def\bbl@toreplace{#5}%
3215 \bbl@TG@@date
3216 \global\bbl@csarg\let{date@\\language @\bbl@tempb}\bbl@toreplace
3217 \ifx\bbl@savestate@empty
3218 \bbl@exp{% TODO. Move to a better place.
3219 \\\AfterBabelCommands{%
3220 \def<\\language date>{\\protect<\\language date >}%
3221 \\\newcommand<\\language date >[4][{}%
3222 \\\bbl@usedategroupttrue
3223 <\bbl@ensure@\\language>{%
3224 \\\localedate[####1]{####2}{####3}{####4}}}%
3225 \def\\bbl@savestate{%
3226 \\\SetString\\today{%
3227 <\\language date>[convert}%
3228 {\the\year}{\the\month}{\the\day}}}%
3229 \fi}%
3230 }}}

```

**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).

```

3231 \let\bbl@calendar@empty
3232 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3233 \@nameuse{\bbl@ca#2}#1@@}
3234 \newcommand\babelDateSpace{\nobreakspace}
3235 \newcommand\babelDateDot{.\@} % TODO. \let instead of repeating
3236 \newcommand\babelDated[1][{\number#1}]
3237 \newcommand\babelDatedd[1][{\ifnum#1<10 0\fi\number#1}]
3238 \newcommand\babelDateM[1][{\number#1}]
3239 \newcommand\babelDateMM[1][{\ifnum#1<10 0\fi\number#1}]
3240 \newcommand\babelDateMMM[1][{%
3241 \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3242 \newcommand\babelDatey[1][{\number#1}]%
3243 \newcommand\babelDateyy[1][{%
3244 \ifnum#1<10 0\number#1 %
3245 \else\ifnum#1<100 \number#1 %
3246 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3247 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3248 \else
3249 \bbl@error{limit-two-digits}{}}}%
3250 \fi\fi\fi}}
3251 \newcommand\babelDateyyyy[1][{\number#1}] % TODO - add leading 0
3252 \newcommand\babelDateU[1][{\number#1}]%
3253 \def\bbl@replace@finish@iii#1{%
3254 \bbl@exp{\def\\#1####1####2####3{\the\toks@}}%
3255 \def\bbl@TG@@date{%
3256 \bbl@replace\bbl@toreplace{[ ]}{\babelDateSpace}}%
3257 \bbl@replace\bbl@toreplace{.}{\babelDateDot}}%
3258 \bbl@replace\bbl@toreplace{[d]}{\babelDated{####3}}%
3259 \bbl@replace\bbl@toreplace{[dd]}{\babelDatedd{####3}}%
3260 \bbl@replace\bbl@toreplace{[M]}{\babelDateM{####2}}%
3261 \bbl@replace\bbl@toreplace{[MM]}{\babelDateMM{####2}}%
3262 \bbl@replace\bbl@toreplace{[MMM]}{\babelDateMMM{####2}}%
3263 \bbl@replace\bbl@toreplace{[y]}{\babelDatey{####1}}%
3264 \bbl@replace\bbl@toreplace{[yy]}{\babelDateyy{####1}}%

```



```

3326 \fi}
3327 \endgroup

```

## 4.18 Handle language system

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

```

3328 \def\bbload@provide@lsys#1{%
3329 \bbload@ifunset{bbload@lname@#1}%
3330 {\bbload@load@info{#1}}%
3331 }%
3332 \bbload@csarg\let{lsys@#1}\empty
3333 \bbload@ifunset{bbload@sname@#1}{\bbload@csarg\gdef{sname@#1}{Default}}}%
3334 \bbload@ifunset{bbload@sotf@#1}{\bbload@csarg\gdef{sotf@#1}{DFLT}}}%
3335 \bbload@csarg\bbload@add@list{lsys@#1}{Script=\bbload@cs{sname@#1}}%
3336 \bbload@ifunset{bbload@lname@#1}{%
3337 {\bbload@csarg\bbload@add@list{lsys@#1}{Language=\bbload@cs{lname@#1}}}%
3338 \ifcase\bbload@engine\or\or
3339 \bbload@ifunset{bbload@prehc@#1}{%
3340 {\bbload@exp{\bbload@ifblank{\bbload@cs{prehc@#1}}}%
3341 }%
3342 {\ifx\bbload@xenohyph\undefined
3343 \global\let\bbload@xenohyph\bbload@xenohyph@d
3344 \ifx\AtBeginDocument\@notprerr
3345 \expandafter\@secondoftwo % to execute right now
3346 \fi
3347 \AtBeginDocument{%
3348 \bbload@patchfont{\bbload@xenohyph}%
3349 {\expandafter\select@language\expandafter{\language}}}%
3350 \fi}}%
3351 \fi
3352 \bbload@csarg\bbload@tglobal{lsys@#1}}
3353 \def\bbload@xenohyph@d{%
3354 \bbload@ifset{bbload@prehc@language}%
3355 {\ifnum\hyphenchar\font=\defaultthyphenchar
3356 \font\bbload@cl{prehc}\relax
3357 \hyphenchar\font\bbload@cl{prehc}\relax
3358 \else\font\bbload@cl{200B}
3359 \hyphenchar\font{200B}
3360 \else
3361 \bbload@warning
3362 {Neither 0 nor ZERO WIDTH SPACE are available\\%
3363 in the current font, and therefore the hyphen\\%
3364 will be printed. Try changing the fontspec's\\%
3365 'HyphenChar' to another value, but be aware\\%
3366 this setting is not safe (see the manual).\\%
3367 Reported}%
3368 \hyphenchar\font\defaultthyphenchar
3369 \fi\fi
3370 \fi}%
3371 {\hyphenchar\font\defaultthyphenchar}}
3372 % \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).

```

3373 \def\bbload@load@ini#1{%
3374 \def\BabelBeforeIni##1##2{%
3375 \begingroup
3376 \bbload@read@ini{##1}0%
3377 \endinput % babel- .tex may contain onlypreamble's
3378 \endgroup}% boxed, to avoid extra spaces:

```



```

3428 \ifcase\@car#8\@nil\or % Currently <10000, but prepared for bigger
3429 \bbl@alphanumeric@ii{#9}000000#1\or
3430 \bbl@alphanumeric@ii{#9}000000#1#2\or
3431 \bbl@alphanumeric@ii{#9}000000#1#2#3\or
3432 \bbl@alphanumeric@ii{#9}000000#1#2#3#4\else
3433 \bbl@alphnum@invalid{>9999}%
3434 \fi}
3435 \def\bbl@alphanumeric@ii#1#2#3#4#5#6#7#8{%
3436 \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\language}%
3437 {\bbl@cs{cntr@#1.4@\language}#5%
3438 \bbl@cs{cntr@#1.3@\language}#6%
3439 \bbl@cs{cntr@#1.2@\language}#7%
3440 \bbl@cs{cntr@#1.1@\language}#8%
3441 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3442 \bbl@ifunset{bbl@cntr@#1.S.321@\language}{}%
3443 {\bbl@cs{cntr@#1.S.321@\language}}%
3444 \fi}%
3445 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}}%
3446 \def\bbl@alphnum@invalid#1{%
3447 \bbl@error{alphabetic-too-large}{#1}{}}

```

## 4.20 Casing

```

3448 \newcommand\BabelUppercaseMapping[3]{%
3449 \DeclareUppercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3450 \newcommand\BabelTitlecaseMapping[3]{%
3451 \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3452 \newcommand\BabelLowercaseMapping[3]{%
3453 \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}

The parser for casing and casing. (variant).
3454 \def\bbl@casemapping#1#2#3{% 1:variant
3455 \def\bbl@tempa##1 ##2{% Loop
3456 \bbl@casemapping@i{##1}%
3457 \ifx\@empty##2\else\bbl@afterfi\bbl@tempa##2\fi}%
3458 \edef\bbl@templ{\@nameuse{bbl@casing@#2}#1}% Language code
3459 \def\bbl@tempe{0}% Mode (upper/lower...)
3460 \def\bbl@tempc{#3}% Casing list
3461 \expandafter\bbl@tempa\bbl@tempc\@empty}
3462 \def\bbl@casemapping@i#1{%
3463 \def\bbl@tempb{#1}%
3464 \ifcase\bbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
3465 \@nameuse{regex_replace_all:nnN}%
3466 {[{\x{c0}-\x{ff}}][{\x{80}-\x{bf}}]*}{\0}}\bbl@tempb
3467 \else
3468 \@nameuse{regex_replace_all:nnN}{.}{\0}}\bbl@tempb % TODO. needed?
3469 \fi
3470 \expandafter\bbl@casemapping@ii\bbl@tempb\@@}
3471 \def\bbl@casemapping@ii#1#2#3\@@{%
3472 \in@{#1#3}{<>}% ie, if <u>, <l>, <t>
3473 \ifin@
3474 \edef\bbl@tempe{%
3475 \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
3476 \else
3477 \ifcase\bbl@tempe\relax
3478 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3479 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#2}}{#1}%
3480 \or
3481 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3482 \or
3483 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3484 \or
3485 \DeclareTitlecaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3486 \fi

```

```
3487 \fi}
```

## 4.21 Getting info

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

```
3488 \def\bbl@localeinfo#1#2{%
3489   \bbl@ifunset{bbl@info@#2}{#1}%
3490   {\bbl@ifunset{bbl@csname bbl@info@#2\endcsname @\languagename}{#1}%
3491    {\bbl@cs{csname bbl@info@#2\endcsname @\languagename}}}%
3492 \newcommand\localeinfo[1]{%
3493   \ifx*#1\empty % TODO. A bit hackish to make it expandable.
3494     \bbl@afterelse\bbl@localeinfo{%
3495       \else
3496         \bbl@localeinfo
3497         {\bbl@error{no-ini-info}{}}{}%
3498         {#1}%
3499       \fi}
3500 % \@namedef{bbl@info@name.locale}{lcname}
3501 \@namedef{bbl@info@tag.ini}{lini}
3502 \@namedef{bbl@info@name.english}{elname}
3503 \@namedef{bbl@info@name.opentype}{lname}
3504 \@namedef{bbl@info@tag.bcp47}{tbc}
3505 \@namedef{bbl@info@language.tag.bcp47}{lbc}
3506 \@namedef{bbl@info@tag.opentype}{lotf}
3507 \@namedef{bbl@info@script.name}{esname}
3508 \@namedef{bbl@info@script.name.opentype}{sname}
3509 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3510 \@namedef{bbl@info@script.tag.opentype}{sotf}
3511 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3512 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3513 \@namedef{bbl@info@extension.t.tag.bcp47}{extt}
3514 \@namedef{bbl@info@extension.u.tag.bcp47}{extu}
3515 \@namedef{bbl@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.

```
3516 \ifcase\bbl@engine % Converts utf8 to its code (expandable)
3517   \def\bbl@utftocode#1{\the\numexpr\decode@UTFviii#1\relax}
3518 \else
3519   \def\bbl@utftocode#1{\expandafter\string#1}
3520 \fi
3521 % Still somewhat hackish. WIP. Note |\str_if_eq:nnTF| is fully
3522 % expandable (|\bbl@ifsamestring| isn't).
3523 \providecommand\BCPdata{}
3524 \ifx\renewcommand\undefined\else % For plain. TODO. It's a quick fix
3525   \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty}
3526   \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3527     \@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3528     {\bbl@bcpdata@ii{#6}\bbl@main@language}%
3529     {\bbl@bcpdata@ii{#1#2#3#4#5#6}\languagename}}%
3530   \def\bbl@bcpdata@ii#1#2{%
3531     \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3532     {\bbl@error{unknown-ini-field}{#1}}{}%
3533     {\bbl@ifunset{bbl@csname bbl@info@#1.tag.bcp47\endcsname @#2}{}%
3534      {\bbl@cs{csname bbl@info@#1.tag.bcp47\endcsname @#2}}}%
3535   \fi
3536 \@namedef{bbl@info@casing.tag.bcp47}{casing}
```

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

```
3537 <<*More package options>> ≡
3538 \DeclareOption{ensureinfo=off}{}
3539 <</More package options>>
```

```

3540 \let\bbl@ensureinfo\@gobble
3541 \newcommand\BabelEnsureInfo{%
3542   \ifx\InputIfFileExists\undefined\else
3543     \def\bbl@ensureinfo##1{%
3544       \bbl@ifunset{\bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3545   \fi
3546   \bbl@foreach\bbl@loaded{%
3547     \let\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files
3548     \def\language{##1}%
3549     \bbl@ensureinfo{##1}}}%
3550 \ifpackagewith{babel}{ensureinfo=off}{}%
3551 {\AtEndOfPackage{% Test for plain.
3552   \ifx\undefined\bbl@loaded\else\BabelEnsureInfo\fi}}

```

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

```

3553 \newcommand\getlocaleproperty{%
3554   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3555 \def\bbl@getproperty@s#1#2#3{%
3556   \let#1\relax
3557   \def\bbl@elt##1##2##3{%
3558     \bbl@ifsamestring{##1/##2}{#3}%
3559     {\providecommand#1{##3}%
3560     \def\bbl@elt###1####2####3{}}}%
3561   {}}%
3562   \bbl@cs{inidata@#2}}%
3563 \def\bbl@getproperty@x#1#2#3{%
3564   \bbl@getproperty@s{#1}{#2}{#3}%
3565   \ifx#1\relax
3566     \bbl@error{unknown-locale-key}{#1}{#2}{#3}%
3567   \fi}
3568 \let\bbl@ini@loaded\@empty
3569 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
3570 \def\ShowLocaleProperties#1{%
3571   \typeout{}}%
3572   \typeout{*** Properties for language '#1' ***}
3573   \def\bbl@elt##1##2##3{\typeout{##1/##2 = ##3}}%
3574   \@nameuse{\bbl@inidata@#1}%
3575   \typeout{*****}}

```

## 5 Adjusting the Babel behavior

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

```

3576 \newcommand\babeladjust[1]{% TODO. Error handling.
3577   \bbl@forkv{#1}{%
3578     \bbl@ifunset{\bbl@ADJ@##1@##2}%
3579     {\bbl@cs{ADJ@##1}{##2}}%
3580     {\bbl@cs{ADJ@##1@##2}}}%
3581   %
3582   \def\bbl@adjust@lua#1#2{%
3583     \ifvmode
3584       \ifnum\currentgrouplevel=\z@
3585         \directlua{ Babel.#2 }%
3586         \expandafter\expandafter\expandafter\@gobble
3587       \fi
3588     \fi
3589     {\bbl@error{adjust-only-vertical}{#1}{}}}% Gobbled if everything went ok.
3590 \namedef{\bbl@ADJ@bidi.mirroring@on}{%
3591   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3592 \namedef{\bbl@ADJ@bidi.mirroring@off}{%
3593   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}

```

```

3594 \@namedef{bbl@ADJ@bidi.text@on}{%
3595   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3596 \@namedef{bbl@ADJ@bidi.text@off}{%
3597   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3598 \@namedef{bbl@ADJ@bidi.math@on}{%
3599   \let\bbl@noamsmath\@empty}
3600 \@namedef{bbl@ADJ@bidi.math@off}{%
3601   \let\bbl@noamsmath\relax}
3602 %
3603 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3604   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3605 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3606   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3607 %
3608 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3609   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3610 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3611   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3612 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3613   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3614 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3615   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3616 \@namedef{bbl@ADJ@justify.arabic@on}{%
3617   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3618 \@namedef{bbl@ADJ@justify.arabic@off}{%
3619   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3620 %
3621 \def\bbl@adjust@layout#1{%
3622   \ifvmode
3623     #1%
3624     \expandafter\@gobble
3625   \fi
3626   {\bbl@error{layout-only-vertical}{}}{}% Gobbled if everything went ok.
3627 \@namedef{bbl@ADJ@layout.tabular@on}{%
3628   \ifnum\bbl@tabular@mode=\tw@
3629     \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}%
3630   \else
3631     \chardef\bbl@tabular@mode\@ne
3632   \fi}
3633 \@namedef{bbl@ADJ@layout.tabular@off}{%
3634   \ifnum\bbl@tabular@mode=\tw@
3635     \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}%
3636   \else
3637     \chardef\bbl@tabular@mode\z@
3638   \fi}
3639 \@namedef{bbl@ADJ@layout.lists@on}{%
3640   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3641 \@namedef{bbl@ADJ@layout.lists@off}{%
3642   \bbl@adjust@layout{\let\list\bbl@OL@list}}
3643 %
3644 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3645   \bbl@bcpallowedtrue}
3646 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3647   \bbl@bcpallowedfalse}
3648 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3649   \def\bbl@bcp@prefix{#1}}
3650 \def\bbl@bcp@prefix{bcp47-}
3651 \@namedef{bbl@ADJ@autoload.options}#1{%
3652   \def\bbl@autoload@options{#1}}
3653 \let\bbl@autoload@bcptoptions\@empty
3654 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3655   \def\bbl@autoload@bcptoptions{#1}}
3656 \newif\ifbbl@bcptname

```



```

3657 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3658   \bbl@bcptonamettrue
3659   \BabelEnsureInfo}
3660 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3661   \bbl@bcptonamefalse}
3662 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3663   \directlua{ Babel.ignore_pre_char = function(node)
3664     return (node.lang == \the\csname l@nohyphenation\endcsname)
3665   end }}
3666 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3667   \directlua{ Babel.ignore_pre_char = function(node)
3668     return false
3669   end }}
3670 \@namedef{bbl@ADJ@interchar.disable@nohyphenation}{%
3671   \def\bbl@ignoreinterchar{%
3672     \ifnum\language=\l@nohyphenation
3673       \expandafter\@gobble
3674     \else
3675       \expandafter\@firstofone
3676     \fi}}
3677 \@namedef{bbl@ADJ@interchar.disable@off}{%
3678   \let\bbl@ignoreinterchar\@firstofone}
3679 \@namedef{bbl@ADJ@select.write@shift}{%
3680   \let\bbl@restorelastskip\relax
3681   \def\bbl@savelastskip{%
3682     \let\bbl@restorelastskip\relax
3683     \ifvmode
3684       \ifdim\lastskip=\z@
3685         \let\bbl@restorelastskip\nobreak
3686       \else
3687         \bbl@exp{%
3688           \def\\bbl@restorelastskip{%
3689             \skip@=\the\lastskip
3690             \\nobreak \vskip-\skip@ \vskip\skip@}}%
3691         \fi
3692       \fi}}
3693 \@namedef{bbl@ADJ@select.write@keep}{%
3694   \let\bbl@restorelastskip\relax
3695   \let\bbl@savelastskip\relax}
3696 \@namedef{bbl@ADJ@select.write@omit}{%
3697   \AddBabelHook{babel-select}{beforestart}{%
3698     \expandafter\babel@aux\expandafter{\bbl@main@language}{}}%
3699   \let\bbl@restorelastskip\relax
3700   \def\bbl@savelastskip##1\bbl@restorelastskip{}}
3701 \@namedef{bbl@ADJ@select.encoding@off}{%
3702   \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.

```

3703 <<{*More package options}>> ≡
3704 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3705 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3706 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}

```

```

3707 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3708 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3709 <</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.

```

3710 \bbl@trace{Cross referencing macros}
3711 \ifx\bbl@opt@safe\empty\else % ie, if 'ref' and/or 'bib'
3712   \def\@newl@bel#1#2#3{%
3713     {\@safe@activestrue
3714       \bbl@ifunset{#1@#2}%
3715       \relax
3716       {\gdef\@multiplelabels{%
3717         \@latex@warning@no@line{There were multiply-defined labels}}%
3718         \@latex@warning@no@line{Label `#2' multiply defined}}%
3719       \global\@namedef{#1@#2}{#3}}}

```

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

```

3720 \CheckCommand*\@testdef[3]{%
3721   \def\reserved@a{#3}%
3722   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3723   \else
3724     \@tempswatrue
3725   \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.

```

3726 \def\@testdef#1#2#3{% TODO. With @samestring?
3727   \@safe@activestrue
3728   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3729   \def\bbl@tempb{#3}%
3730   \@safe@activesfalse
3731   \ifx\bbl@tempa\relax
3732   \else
3733     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3734   \fi
3735   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3736   \ifx\bbl@tempa\bbl@tempb
3737   \else
3738     \@tempswatrue
3739   \fi}
3740 \fi

```

## **\ref**

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

```

3741 \bbl@xin@{R}\bbl@opt@safe
3742 \ifin@
3743   \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3744   \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3745   {\expandafter\strip@prefix\meaning\ref}%
3746 \ifin@
3747   \bbl@redefine\@kernel@ref#1{%
3748     \@safe@activestrue\org@@kernel@ref{#1}\@safe@activesfalse}

```

```

3749 \bbl@redefine\@kernel@pageref#1{%
3750 \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activfalse}
3751 \bbl@redefine\@kernel@sref#1{%
3752 \@safe@activetrue\org@@kernel@sref{#1}\@safe@activfalse}
3753 \bbl@redefine\@kernel@spageref#1{%
3754 \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activfalse}
3755 \else
3756 \bbl@redefineroast\ref#1{%
3757 \@safe@activetrue\org@ref{#1}\@safe@activfalse}
3758 \bbl@redefineroast\pageref#1{%
3759 \@safe@activetrue\org@pageref{#1}\@safe@activfalse}
3760 \fi
3761 \else
3762 \let\org@ref\ref
3763 \let\org@pageref\pageref
3764 \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.

```

3765 \bbl@xin@{B}\bbl@opt@safe
3766 \ifin@
3767 \bbl@redefine\@citex[#1]#2{%
3768 \@safe@activetrue\edef\bbl@tempa{#2}\@safe@activfalse
3769 \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.

```

3770 \AtBeginDocument{%
3771 \ifpackageloaded{natbib}{%

```

Notice that we use \def here instead of \bbl@redefine because \org@@citex is already defined and we don't want to overwrite that definition (it would result in parameter stack overflow because of a circular definition).

(Recent versions of natbib change dynamically \@citex, so PR4087 doesn't seem fixable in a simple way. Just load natbib before.)

```

3772 \def\@citex[#1][#2]#3{%
3773 \@safe@activetrue\edef\bbl@tempa{#3}\@safe@activfalse
3774 \org@@citex[#1][#2]{\bbl@tempa}}%
3775 }{}

```

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

```

3776 \AtBeginDocument{%
3777 \ifpackageloaded{cite}{%
3778 \def\@citex[#1]#2{%
3779 \@safe@activetrue\org@@citex[#1][#2]\@safe@activfalse}%
3780 }{}

```

**\nocite** The macro \nocite which is used to instruct BiB<sub>T</sub><sub>E</sub>X to extract uncited references from the database.

```

3781 \bbl@redefine\nocite#1{%
3782 \@safe@activetrue\org@nocite{#1}\@safe@activfalse}

```

**\bibcite** The macro that is used in the .aux file to define citation labels. When packages such as natbib or cite are not loaded its second argument is used to typeset the citation label. In that case, this second argument can contain active characters but is used in an environment where \@safe@activetrue is in effect. This switch needs to be reset inside the \hbox which contains the citation label. In order to determine during .aux file processing which definition of \bibcite is

needed we define `\bibcite` in such a way that it redefines itself with the proper definition. We call `\bbl@cite@choice` to select the proper definition for `\bibcite`. This new definition is then activated.

```
3783 \bbl@redefine\bibcite{%
3784   \bbl@cite@choice
3785   \bibcite}
```

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

```
3786 \def\bbl@bibcite#1#2{%
3787   \org@bibcite{#1}{\@safe@activesfalse#2}}
```

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

```
3788 \def\bbl@cite@choice{%
3789   \global\let\bibcite\bbl@bibcite
3790   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
3791   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
3792   \global\let\bbl@cite@choice\relax}
```

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

```
3793 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3794 \bbl@redefine\@bibitem#1{%
3795   \@safe@activetrue\org@@bibitem{#1}\@safe@activesfalse}
3796 \else
3797   \let\org@nocite\nocite
3798   \let\org@@citex\citex
3799   \let\org@bibcite\bibcite
3800   \let\org@@bibitem\@bibitem
3801 \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.

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

```

3820      \bbl@exp{%
3821          \\org@markright{\\protect\\foreignlanguage{\language}%
3822              {\\protect\\bbl@restore@actives\the\toks@}}}%

```

## **\markboth**

**\@mkboth** 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.)

```

3823      \ifx\@mkboth\markboth
3824      \def\bbl@tempc{\let\@mkboth\markboth}%
3825      \else
3826      \def\bbl@tempc{%
3827      \fi
3828      \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
3829      \markboth#1#2{%
3830          \protected@edef\bbl@tempb##1{%
3831              \protect\foreignlanguage
3832              {\language}{\protect\bbl@restore@actives##1}}%
3833          \bbl@ifblank{#1}%
3834              {\toks@}%
3835              {\toks@\expandafter{\bbl@tempb{#1}}}%
3836          \bbl@ifblank{#2}%
3837              {\@temptokena}%
3838              {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3839          \bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}}%
3840          \bbl@tempc
3841      \fi} % end ifbbl@single, end \IfBabelLayout

```

## 5.3 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.

```

3842 \bbl@trace{Preventing clashes with other packages}
3843 \ifx\org@ref\undefined\else
3844     \bbl@xin@{R}\bbl@opt@safe
3845     \ifin@
3846         \AtBeginDocument{%
3847             \@ifpackageloaded{ifthen}{%
3848                 \bbl@redefine@long\ifthenelse#1#2#3{%
3849                     \let\bbl@temp@pref\pageref
3850                     \let\pageref\org@pageref

```

```

3851      \let\bbl@temp@ref\ref
3852      \let\ref\org@ref
3853      \@safe@activestrue
3854      \org@ifthenelse{#1}%
3855        {\let\pageref\bbl@temp@pref
3856         \let\ref\bbl@temp@ref
3857         \@safe@activesfalse
3858         #2}%
3859        {\let\pageref\bbl@temp@pref
3860         \let\ref\bbl@temp@ref
3861         \@safe@activesfalse
3862         #3}%
3863      }%
3864    }{}%
3865  }
3866 \fi

```

### 5.3.2 varioref

**\@@vpageref**

**\vrefpagemum**

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

```

3867 \AtBeginDocument{%
3868   \ifpackageloaded{varioref}{%
3869     \bbl@redefine\@@vpageref#1[#2]#3{%
3870       \@safe@activestrue
3871       \org@@@vpageref{#1}[#2]{#3}%
3872       \@safe@activesfalse}%
3873     \bbl@redefine\vrefpagemum#1#2{%
3874       \@safe@activestrue
3875       \org@vrefpagemum{#1}{#2}%
3876       \@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.

```

3877   \expandafter\def\csname Ref \endcsname#1{%
3878     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3879   }{}%
3880 }
3881 \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.

```

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

```

```

3889     \fi}%
3890     {}}}

```

**\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.

```

3891 \def\substitutefontfamily#1#2#3{%
3892   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3893   \immediate\write15{%
3894     \string\ProvidesFile{#1#2.fd}%
3895     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3896     \space generated font description file]^J
3897     \string\DeclareFontFamily{#1}{#2}{}}^J
3898     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}}^J
3899     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}}^J
3900     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}}^J
3901     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}}^J
3902     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}}^J
3903     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}}^J
3904     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}}^J
3905     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}}^J
3906   }%
3907   \closeout15
3908 }
3909 \@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**

```

3910 \bbl@trace{Encoding and fonts}
3911 \newcommand\BabelNonASCII{LGR,LGI,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3912 \newcommand\BabelNonText{TS1,T3,TS3}
3913 \let\org@TeX\TeX
3914 \let\org@LaTeX\LaTeX
3915 \let\ensureascii@firstofone
3916 \let\asciientcoding\@empty
3917 \AtBeginDocument{%
3918   \def\elt#1{,#1,}%
3919   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3920   \let\elt\relax
3921   \let\bbl@tempb\@empty
3922   \def\bbl@tempc{OT1}%
3923   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3924     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3925   \bbl@foreach\bbl@tempa{%
3926     \bbl@xin@{,#1,},{,\BabelNonASCII,}%
3927     \ifin@
3928       \def\bbl@tempb{#1}% Store last non-ascii
3929     \else\bbl@xin@{,#1,},{,\BabelNonText,}% Pass
3930       \ifin@
3931         \def\bbl@tempc{#1}% Store last ascii
3932       \fi
3933     \fi}%
3934   \ifx\bbl@tempb\@empty\else
3935     \bbl@xin@{,\cf@encoding,},{,\BabelNonASCII,\BabelNonText,}%

```

```

3936 \ifin@else
3937 \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3938 \fi
3939 \let\asciencoding\bbl@tempc
3940 \renewcommand\ensureasci[1]{%
3941 {\fontencoding\asciencoding}\selectfont#1}}%
3942 \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3943 \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3944 \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.

```

3945 \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.

```

3946 \AtBeginDocument{%
3947 \ifpackageloaded{fontspec}%
3948 {\xdef\latinencoding{%
3949 \ifx\UTFencname\undefined
3950 EU\ifcase\bbl@engine\or2\or1\fi
3951 \else
3952 \UTFencname
3953 \fi}}%
3954 {\gdef\latinencoding{OT1}%
3955 \ifx\cf@encoding\bbl@t@one
3956 \xdef\latinencoding{\bbl@t@one}%
3957 \else
3958 \def\@elt#1{,#1,}%
3959 \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3960 \let\@elt\relax
3961 \bbl@xin@{,T1,}\bbl@tempa
3962 \ifin@
3963 \xdef\latinencoding{\bbl@t@one}%
3964 \fi
3965 \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.

```

3966 \DeclareRobustCommand{\latintext}{%
3967 \fontencoding{\latinencoding}\selectfont
3968 \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.

```

3969 \ifx\@undefined\DeclareTextFontCommand
3970 \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3971 \else
3972 \DeclareTextFontCommand{\textlatin}{\latintext}
3973 \fi

```

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

```

3974 \def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}

```



## 5.5 Basic bidi support

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.

```
3975 \bbl@trace{Loading basic (internal) bidi support}
3976 \ifodd\bbl@engine
3977 \else % TODO. Move to txtbabel. Any xe+lua bidi
3978   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3979     \bbl@error{bidi-only-lua}{\}\}\}%
3980     \let\bbl@beforeforeign\leavevmode
3981     \AtEndOfPackage{%
3982       \EnableBabelHook{babel-bidi}%
3983       \bbl@xebidipar}
3984 \fi\fi
3985 \def\bbl@loadxebidi#1{%
3986   \ifx\RTLfootnotetext\@undefined
3987     \AtEndOfPackage{%
3988       \EnableBabelHook{babel-bidi}%
3989       \ifx\fontspec\@undefined
3990         \usepackage{fontspec}% bidi needs fontspec
3991       \fi
3992       \usepackage#1{bidi}%
3993       \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
3994       \def\DigitsDotDashInterCharToks{% See the 'bidi' package
3995         \ifnum\@nameuse{\bbl@wdir\@languagename}=\tw@ % 'AL' bidi
3996           \bbl@digitsdotdash % So ignore in 'R' bidi
3997         \fi}}%
3998   \fi}
3999 \ifnum\bbl@bidimode>200 % Any xe bidi=
4000   \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
4001     \bbl@tentative{bidi=bidi}
4002     \bbl@loadxebidi{}
4003   \or
4004     \bbl@loadxebidi{[rldocument]}
4005   \or
4006     \bbl@loadxebidi{}
4007   \fi
4008 \fi
4009 \fi
4010 % TODO? Separate:
4011 \ifnum\bbl@bidimode=\@ne % bidi=default
4012   \let\bbl@beforeforeign\leavevmode
4013   \ifodd\bbl@engine % lua
4014     \newattribute\bbl@attr@dir
4015     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
```

```

4016 \bbl@exp{\output{\bodydir\pagedir\the\output}}
4017 \fi
4018 \AtEndOfPackage{%
4019 \EnableBabelHook{babel-bidi}% pdf/luax/xe
4020 \ifodd\bbl@engine\else % pdf/xe
4021 \bbl@xebidipar
4022 \fi}
4023 \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.

```

4024 \bbl@trace{Macros to switch the text direction}
4025 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
4026 \def\bbl@rscripts{%
4027 ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
4028 Old Hungarian,Lydian,Mandaean,Manichaean,%
4029 Meroitic Cursive,Meroitic,Old North Arabian,%
4030 Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
4031 Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
4032 Old South Arabian,}%
4033 \def\bbl@provide@dirs#1{%
4034 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
4035 \ifin@
4036 \global\bbl@csarg\chardef{wdir@#1}\@ne
4037 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4038 \ifin@
4039 \global\bbl@csarg\chardef{wdir@#1}\tw@
4040 \fi
4041 \else
4042 \global\bbl@csarg\chardef{wdir@#1}\z@
4043 \fi
4044 \ifodd\bbl@engine
4045 \bbl@csarg\ifcase{wdir@#1}%
4046 \directlua{ Babel.locale_props[\the\localeid].texdir = 'l' }%
4047 \or
4048 \directlua{ Babel.locale_props[\the\localeid].texdir = 'r' }%
4049 \or
4050 \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
4051 \fi
4052 \fi}
4053 \def\bbl@switchdir{%
4054 \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4055 \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
4056 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}%
4057 \def\bbl@setdirs#1{% TODO - math
4058 \ifcase\bbl@select@type % TODO - strictly, not the right test
4059 \bbl@bodydir{#1}%
4060 \bbl@pardir{#1}% <- Must precede \bbl@texdir
4061 \fi
4062 \bbl@texdir{#1}}
4063 \ifnum\bbl@bidimode>\z@
4064 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4065 \DisableBabelHook{babel-bidi}
4066 \fi

```

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

```

4067 \ifodd\bbl@engine % luatex=1
4068 \else % pdftex=0, xetex=2
4069 \newcount\bbl@dirlevel
4070 \chardef\bbl@thetexdir\z@
4071 \chardef\bbl@thepardir\z@
4072 \def\bbl@texdir#1{%
4073 \ifcase#1\relax

```

```

4074     \chardef\bbl@thetextdir\z@
4075     \@nameuse{setlatin}%
4076     \bbl@textdir@i\beginL\endL
4077   \else
4078     \chardef\bbl@thetextdir\@ne
4079     \@nameuse{setnonlatin}%
4080     \bbl@textdir@i\beginR\endR
4081   \fi}
4082 \def\bbl@textdir@i#1#2{%
4083   \ifhmode
4084     \ifnum\currentgrouplevel>\z@
4085       \ifnum\currentgrouplevel=\bbl@dirlevel
4086         \bbl@error{multiple-bidi}{\}\}\}%
4087         \bgroup\aftergroup#2\aftergroup\egroup
4088       \else
4089         \ifcase\currentgrouptype\or % 0 bottom
4090           \aftergroup#2% 1 simple {}
4091         \or
4092           \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4093         \or
4094           \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4095         \or\or\or % vbox vtop align
4096         \or
4097           \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4098         \or\or\or\or\or\or\or % output math disc insert vcent mathchoice
4099         \or
4100           \aftergroup#2% 14 \begingroup
4101         \else
4102           \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4103         \fi
4104       \fi
4105       \bbl@dirlevel\currentgrouplevel
4106     \fi
4107     #1%
4108   \fi}
4109 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4110 \let\bbl@bodydir\@gobble
4111 \let\bbl@pagedir\@gobble
4112 \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).

```

4113 \def\bbl@xebidipar{%
4114   \let\bbl@xebidipar\relax
4115   \TeXeTstate\@ne
4116   \def\bbl@xeeverypar{%
4117     \ifcase\bbl@thepardir
4118       \ifcase\bbl@thetextdir\else\beginR\fi
4119     \else
4120       {\setbox\z@\lastbox\beginR\box\z@}%
4121     \fi}%
4122   \AddToHook{para/begin}{\bbl@xeeverypar}}
4123 \ifnum\bbl@bidimode>200 % Any xe bidi=
4124   \let\bbl@textdir@i\@gobbletwo
4125   \let\bbl@xebidipar\@empty
4126   \AddBabelHook{bidi}{foreign}{%
4127     \ifcase\bbl@thetextdir
4128       \BabelWrapText{\LR{##1}}%
4129     \else
4130       \BabelWrapText{\RL{##1}}%
4131     \fi}
4132   \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}

```

```

4133 \fi
4134 \fi

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

4135 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4136 \AtBeginDocument{%
4137   \ifx\pdfstringdefDisableCommands\undefined\else
4138     \ifx\pdfstringdefDisableCommands\relax\else
4139       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4140     \fi
4141   \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.

```

4142 \bbl@trace{Local Language Configuration}
4143 \ifx\loadlocalcfg\undefined
4144   \ifpackagewith{babel}{noconfigs}%
4145     {\let\loadlocalcfg\@gobble}%
4146     {\def\loadlocalcfg#1{%
4147       \InputIfFileExists{#1.cfg}%
4148       {\typeout{*****^J%
4149                 * Local config file #1.cfg used^^J%
4150                 *}}%
4151       \@empty}}
4152 \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).

```

4153 \bbl@trace{Language options}
4154 \let\bbl@afterlang\relax
4155 \let\BabelModifiers\relax
4156 \let\bbl@loaded\@empty
4157 \def\bbl@load@language#1{%
4158   \InputIfFileExists{#1.ldf}%
4159   {\edef\bbl@loaded{\CurrentOption
4160     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4161     \expandafter\let\expandafter\bbl@afterlang
4162       \csname\CurrentOption.ldf-h@k\endcsname
4163     \expandafter\let\expandafter\BabelModifiers
4164       \csname bbl@mod@\CurrentOption\endcsname
4165     \bbl@exp{\AtBeginDocument{%
4166       \bbl@usehooks@lang{\CurrentOption}{begindocument}{\CurrentOption}}}%
4167     {\IfFileExists{babel-#1.tex}%
4168      {\def\bbl@tempa{%
4169        .\\There is a locale ini file for this language.\\%
4170        If it's the main language, try adding `provide=*'\%
4171        to the babel package options}}%
4172      {\let\bbl@tempa\empty}%
4173      \bbl@error{unknown-package-option}{\CurrentOption}}}

```

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

```

4174 \def\bbl@try@load@lang#1#2#3{%
4175   \IfFileExists{\CurrentOption.ldf}%

```

```

4176 {\bbl@load@language{\CurrentOption}}%
4177 {\#1\bbl@load@language{\#2}\#3}}
4178 %
4179 \DeclareOption{hebrew}{%
4180 \ifcase\bbl@engine\or
4181 \bbl@error{only-pdftex-lang}{hebrew}{luatex}}}%
4182 \fi
4183 \input{rlbabel.def}%
4184 \bbl@load@language{hebrew}}
4185 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4186 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4187 \DeclareOption{polutonikogreek}{%
4188 \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4189 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4190 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4191 \DeclareOption{uppersorbian}{\bbl@try@load@lang{}{usorbian}{}}

```

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

```

4192 \ifx\bbl@opt@config\@nnil
4193 \@ifpackagewith{babel}{noconfigs}}}%
4194 {\InputIfFileExists{bblopts.cfg}%
4195 {\typeout{*****^J%
4196 * Local config file bblopts.cfg used^J%
4197 *}}}%
4198 {}}%
4199 \else
4200 \InputIfFileExists{\bbl@opt@config.cfg}%
4201 {\typeout{*****^J%
4202 * Local config file \bbl@opt@config.cfg used^J%
4203 *}}}%
4204 {\bbl@error{config-not-found}}{}{}%
4205 \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.

```

4206 \ifx\bbl@opt@main\@nnil
4207 \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4208 \let\bbl@tempb\@empty
4209 \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4210 \bbl@foreach\bbl@tempa{\edef\bbl@tempb{\#1,\bbl@tempb}}}%
4211 \bbl@foreach\bbl@tempb{% \bbl@tempb is a reversed list
4212 \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4213 \ifodd\bbl@iniflag % = *
4214 \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}}%
4215 \else % n +=
4216 \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}}%
4217 \fi
4218 \fi}%
4219 \fi
4220 \else
4221 \bbl@info{Main language set with 'main='. Except if you have\\%
4222 problems, prefer the default mechanism for setting\\%
4223 the main language, ie, as the last declared.\\%
4224 Reported}
4225 \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`).

```
4226 \ifx\bbbl@opt@main\@nnil\else
4227   \bbbl@ncarg\let\bbbl@loadmain{ds@\bbbl@opt@main}%
4228   \expandafter\let\csname ds@\bbbl@opt@main\endcsname\relax
4229 \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.

```
4230 \bbbl@foreach\bbbl@language@opts{%
4231   \def\bbbl@tempa{#1}%
4232   \ifx\bbbl@tempa\bbbl@opt@main\else
4233     \ifnum\bbbl@iniflag<\tw@ % 0 0 (other = ldf)
4234       \bbbl@ifunset{ds@#1}%
4235       {\DeclareOption{#1}{\bbbl@load@language{#1}}}%
4236       {}%
4237     \else % + * (other = ini)
4238       \DeclareOption{#1}{%
4239         \bbbl@ldfinit
4240         \babelprovide[import]{#1}%
4241         \bbbl@afterldf{}}%
4242     \fi
4243   \fi}
4244 \bbbl@foreach\@classoptionslist{%
4245   \def\bbbl@tempa{#1}%
4246   \ifx\bbbl@tempa\bbbl@opt@main\else
4247     \ifnum\bbbl@iniflag<\tw@ % 0 0 (other = ldf)
4248       \bbbl@ifunset{ds@#1}%
4249       {\IfFileExists{#1.ldf}%
4250        {\DeclareOption{#1}{\bbbl@load@language{#1}}}%
4251        {}}%
4252       {}%
4253     \else % + * (other = ini)
4254       \IfFileExists{babel-#1.tex}%
4255       {\DeclareOption{#1}{%
4256         \bbbl@ldfinit
4257         \babelprovide[import]{#1}%
4258         \bbbl@afterldf{}}}%
4259       {}%
4260     \fi
4261   \fi}
```

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

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

```
4262 \def\AfterBabelLanguage#1{%
4263   \bbbl@ifsamestring\CurrentOption{#1}{\global\bbbl@add\bbbl@afterlang}{}}
4264 \DeclareOption*{}
4265 \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`.

```
4266 \bbbl@trace{Option 'main'}
4267 \ifx\bbbl@opt@main\@nnil
4268   \edef\bbbl@tempa{\@classoptionslist,\bbbl@language@opts}
4269   \let\bbbl@tempc\empty
4270   \edef\bbbl@templ{,\bbbl@loaded,}
4271   \edef\bbbl@templ{\expandafter\strip@prefix\meaning\bbbl@templ}
```

```

4272 \bbl@for\bbl@tempb\bbl@tempa{%
4273   \edef\bbl@tempd{,\bbl@tempb,}%
4274   \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4275   \bbl@xin@{\bbl@tempd}{\bbl@tempb}%
4276   \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
4277 \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4278 \expandafter\bbl@tempa\bbl@loaded,\@nnil
4279 \ifx\bbl@tempb\bbl@tempc\else
4280   \bbl@warning{%
4281     Last declared language option is '\bbl@tempc',\%
4282     but the last processed one was '\bbl@tempb'.\%
4283     The main language can't be set as both a global\%
4284     and a package option. Use 'main=\bbl@tempc' as\%
4285     option. Reported}
4286 \fi
4287 \else
4288   \ifodd\bbl@iniflag % case 1,3 (main is ini)
4289     \bbl@ldfinit
4290     \let\CurrentOption\bbl@opt@main
4291     \bbl@exp{% \bbl@opt@provide = empty if *
4292       \\\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4293     \bbl@afterldf{}
4294     \DeclareOption{\bbl@opt@main}{}
4295   \else % case 0,2 (main is ldf)
4296     \ifx\bbl@loadmain\relax
4297       \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4298     \else
4299       \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4300     \fi
4301     \ExecuteOptions{\bbl@opt@main}
4302     \@namedef{ds@\bbl@opt@main}{}%
4303   \fi
4304   \DeclareOption*{}
4305   \ProcessOptions*
4306 \fi
4307 \bbl@exp{%
4308   \\\AtBeginDocument{\\\bbl@usehooks@lang{/}{\begindocument}{}}}%
4309 \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.

```

4310 \ifx\bbl@main@language\@undefined
4311   \bbl@info{%
4312     You haven't specified a language as a class or package\%
4313     option. I'll load 'nil'. Reported}
4314   \bbl@load@language{nil}
4315 \fi
4316 \end{package}

```

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

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

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

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

A proxy file for `switch.def`

```

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

```

## 7 Error messages

They are loaded when `\bbl@error` is first called. To save space, the main code just identifies them with a tag, and messages are stored in a separate file. Since it can be loaded anywhere, you make sure some catcodes have the right value, although those for `\`, ```, `^M`, `%` and `=` are reset before loading the file.

```

4322 <{*errors}>
4323 \catcode`\{=1 \catcode`\}=2 \catcode`\#=6
4324 \catcode`\:=12 \catcode`\,=12 \catcode`\.=12 \catcode`\-=12
4325 \catcode`\'=12 \catcode`\(=12 \catcode`\)=12
4326 \catcode`\@=11 \catcode`\^=7
4327 %
4328 \ifx\MessageBreak\@undefined
4329 \gdef\bbl@error@i#1#2{%
4330 \beginingroup
4331 \newlinechar=`^^J
4332 \def\{^^J(babel) }%
4333 \errhelp{#2}\errmessage{\{#1}%
4334 \endgroup}
4335 \else
4336 \gdef\bbl@error@i#1#2{%
4337 \beginingroup
4338 \def\{\MessageBreak}%
4339 \PackageError{babel}{#1}{#2}%
4340 \endgroup}
4341 \fi
4342 \def\bbl@errmessage#1#2#3{%
4343 \expandafter\gdef\csname bbl@err@#1\endcsname##1##2##3{%
4344 \bbl@error@i{#2}{#3}}
4345 % Implicit #2#3#4:
4346 \gdef\bbl@error#1{\csname bbl@err@#1\endcsname}
4347 %
4348 \bbl@errmessage{not-yet-available}
4349 {Not yet available}%
4350 {Find an armchair, sit down and wait}
4351 \bbl@errmessage{bad-package-option}%
4352 {Bad option '#1=#2'. Either you have misspelled the\\%
4353 key or there is a previous setting of '#1'. Valid\\%
4354 keys are, among others, 'shorthands', 'main', 'bidi',\\%
4355 'strings', 'config', 'headfoot', 'safe', 'math'.}%
4356 {See the manual for further details.}
4357 \bbl@errmessage{base-on-the-fly}
4358 {For a language to be defined on the fly 'base'\\%
4359 is not enough, and the whole package must be\\%
4360 loaded. Either delete the 'base' option or\\%
4361 request the languages explicitly}%
4362 {See the manual for further details.}
4363 \bbl@errmessage{undefined-language}
4364 {You haven't defined the language '#1' yet.\\%
4365 Perhaps you misspelled it or your installation\\%
4366 is not complete}%
4367 {Your command will be ignored, type <return> to proceed}
4368 \bbl@errmessage{shorthand-is-off}
4369 {I can't declare a shorthand turned off (\string#2)}
4370 {Sorry, but you can't use shorthands which have been\\%
4371 turned off in the package options}

```



```

4372 \bbl@errmessage{not-a-shorthand}
4373 {The character '\string #1' should be made a shorthand character;\%
4374   add the command \string\useshorthands\string{#1\string} to
4375   the preamble.\%
4376   I will ignore your instruction}%
4377 {You may proceed, but expect unexpected results}
4378 \bbl@errmessage{not-a-shorthand-b}
4379 {I can't switch '\string#2' on or off--not a shorthand}%
4380 {This character is not a shorthand. Maybe you made\%
4381   a typing mistake? I will ignore your instruction.}
4382 \bbl@errmessage{unknown-attribute}
4383 {The attribute #2 is unknown for language #1.}%
4384 {Your command will be ignored, type <return> to proceed}
4385 \bbl@errmessage{missing-group}
4386 {Missing group for string \string#1}%
4387 {You must assign strings to some category, typically\%
4388   captions or extras, but you set none}
4389 \bbl@errmessage{only-lua-xe}
4390 {This macro is available only in LuaLaTeX and XeLaTeX.}%
4391 {Consider switching to these engines.}
4392 \bbl@errmessage{only-lua}
4393 {This macro is available only in LuaLaTeX}%
4394 {Consider switching to that engine.}
4395 \bbl@errmessage{unknown-provide-key}
4396 {Unknown key '#1' in \string\babelprovide}%
4397 {See the manual for valid keys}%
4398 \bbl@errmessage{unknown-mapfont}
4399 {Option '\bbl@KVP@mapfont' unknown for\%
4400   mapfont. Use 'direction'}%
4401 {See the manual for details.}
4402 \bbl@errmessage{no-ini-file}
4403 {There is no ini file for the requested language\%
4404   (#1: \languagename). Perhaps you misspelled it or your\%
4405   installation is not complete}%
4406 {Fix the name or reinstall babel.}
4407 \bbl@errmessage{digits-is-reserved}
4408 {The counter name 'digits' is reserved for mapping\%
4409   decimal digits}%
4410 {Use another name.}
4411 \bbl@errmessage{limit-two-digits}
4412 {Currently two-digit years are restricted to the\%
4413   range 0-9999}%
4414 {There is little you can do. Sorry.}
4415 \bbl@errmessage{alphabetic-too-large}
4416 {Alphabetic numeral too large (#1)}%
4417 {Currently this is the limit.}
4418 \bbl@errmessage{no-ini-info}
4419 {I've found no info for the current locale.\%
4420   The corresponding ini file has not been loaded\%
4421   Perhaps it doesn't exist}%
4422 {See the manual for details.}
4423 \bbl@errmessage{unknown-ini-field}
4424 {Unknown field '#1' in \string\BCPdata.\%
4425   Perhaps you misspelled it}%
4426 {See the manual for details.}
4427 \bbl@errmessage{unknown-locale-key}
4428 {Unknown key for locale '#2':\%
4429   #3\%
4430   \string#1 will be set to \string\relax}%
4431 {Perhaps you misspelled it.}%
4432 \bbl@errmessage{adjust-only-vertical}
4433 {Currently, #1 related features can be adjusted only\%
4434   in the main vertical list}%

```

```

4435 {Maybe things change in the future, but this is what it is.}
4436 \bbl@errmessage{layout-only-vertical}
4437 {Currently, layout related features can be adjusted only\\%
4438   in vertical mode}%
4439 {Maybe things change in the future, but this is what it is.}
4440 \bbl@errmessage{bidi-only-lua}
4441 {The bidi method 'basic' is available only in\\%
4442   luatex. I'll continue with 'bidi=default', so\\%
4443   expect wrong results}%
4444 {See the manual for further details.}
4445 \bbl@errmessage{multiple-bidi}
4446 {Multiple bidi settings inside a group}%
4447 {I'll insert a new group, but expect wrong results.}
4448 \bbl@errmessage{unknown-package-option}
4449 {Unknown option '\CurrentOption'. Either you misspelled it\\%
4450   or the language definition file \CurrentOption.ldf\\%
4451   was not found%
4452   \bbl@tempa}
4453 {Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4454   activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4455   headfoot=, strings=, config=, hyphenmap=, or a language name.}
4456 \bbl@errmessage{config-not-found}
4457 {Local config file '\bbl@opt@config.cfg' not found}%
4458 {Perhaps you misspelled it.}
4459 \bbl@errmessage{late-after-babel}
4460 {Too late for \string\AfterBabelLanguage}%
4461 {Languages have been loaded, so I can do nothing}
4462 \bbl@errmessage{double-hyphens-class}
4463 {Double hyphens aren't allowed in \string\babelcharclass\\%
4464   because it's potentially ambiguous}%
4465 {See the manual for further info}
4466 \bbl@errmessage{unknown-interchar}
4467 {'#1' for '\language' cannot be enabled.\\%
4468   Maybe there is a typo}%
4469 {See the manual for further details.}
4470 \bbl@errmessage{unknown-interchar-b}
4471 {'#1' for '\language' cannot be disabled.\\%
4472   Maybe there is a typo}%
4473 {See the manual for further details.}
4474 \bbl@errmessage{charproperty-only-vertical}
4475 {\string\babelcharproperty\space can be used only in\\%
4476   vertical mode (preamble or between paragraphs)}%
4477 {See the manual for further info}
4478 \bbl@errmessage{unknown-char-property}
4479 {No property named '#2'. Allowed values are\\%
4480   direction (bc), mirror (bmg), and linebreak (lb)}%
4481 {See the manual for further info}
4482 \bbl@errmessage{bad-transform-option}
4483 {Bad option '#1' in a transform.\\%
4484   I'll ignore it but expect more errors}%
4485 {See the manual for further info.}
4486 \bbl@errmessage{font-conflict-transforms}
4487 {Transforms cannot be re-assigned to different\\%
4488   fonts. The conflict is in '\bbl@kv@label'.\\%
4489   Apply the same fonts or use a different label}%
4490 {See the manual for further details.}
4491 \bbl@errmessage{transform-not-available}
4492 {'#1' for '\language' cannot be enabled.\\%
4493   Maybe there is a typo or it's a font-dependent transform}%
4494 {See the manual for further details.}
4495 \bbl@errmessage{transform-not-available-b}
4496 {'#1' for '\language' cannot be disabled.\\%
4497   Maybe there is a typo or it's a font-dependent transform}%

```

```

4498 {See the manual for further details.}
4499 \bbl@errmessage{year-out-range}
4500 {Year out of range.\\%
4501   The allowed range is #1}%
4502 {See the manual for further details.}
4503 \bbl@errmessage{only-pdftex-lang}
4504 {The '#1' ldf style doesn't work with #2,\\%
4505   but you can use the ini locale instead.\\%
4506   Try adding 'provide=*' to the option list. You may\\%
4507   also want to set 'bidi=' to some value}%
4508 {See the manual for further details.}
4509 \bbl@errmessage{hyphenmins-args}
4510 {\string\babelhyphenmins\ accepts either the optional\\%
4511   argument or the star, but not both at the same time}%
4512 {See the manual for further details.}
4513 </errors>
4514 <*patterns>

```

## 8 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.

```

4515 <@Make sure ProvidesFile is defined@>
4516 \ProvidesFile{hyphen.cfg}[<@date@> v<@version@> Babel hyphens]
4517 \xdef\bbl@format{\jobname}
4518 \def\bbl@version{<@version@>}
4519 \def\bbl@date{<@date@>}
4520 \ifx\AtBeginDocument\@undefined
4521   \def\@empty{}
4522 \fi
4523 <@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.

```

4524 \def\process@line#1#2 #3 #4 {%
4525   \ifx=#1%
4526     \process@synonym{#2}%
4527   \else
4528     \process@language{#1#2}{#3}{#4}%
4529   \fi
4530   \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.

```

4531 \toks@{}
4532 \def\bbl@languages{}

```

When no languages have been loaded yet, the name following the `=` will be a synonym for hyphenation register 0. So, it is stored in a token register and executed when the first pattern file has been processed. (The `\relax` just helps to the `\if` below catching synonyms without a language.) Otherwise the name will be a synonym for the language loaded last. We also need to copy the `hyphenmin` parameters for the synonym.

```

4533 \def\process@synonym#1{%
4534   \ifnum\last@language=@m@ne
4535     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4536   \else
4537     \expandafter\chardef\csname l@#1\endcsname\last@language
4538     \wlog{\string\l@#1=\string\language\the\last@language}%
4539     \expandafter\let\csname #1hyphenmins\expandafter\endcsname

```

```

4540 \csname\language hyphenmins\endcsname
4541 \let\bbl@elt\relax
4542 \edef\bbl@languages{\bbl@languages\bbl@elt{#1}\the\last@language}{}}}%
4543 \fi}

```

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

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

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

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

Pattern files may contain assignments to `\lefthyphenmin` and `\righthyphenmin`.  $\TeX$  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 en \uccode arrays`. Such changes should remain local to the language; therefore we process the pattern file in a group; the `\patterns` command acts globally so its effect will be remembered.

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

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

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

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

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

```

4544 \def\process@language#1#2#3{%
4545 \expandafter\addlanguage\csname l@#1\endcsname
4546 \expandafter\language\csname l@#1\endcsname
4547 \edef\language{#1}%
4548 \bbl@hook@everylanguage{#1}%
4549 % > luatex
4550 \bbl@get@enc#1:.\@@@
4551 \begingroup
4552 \lefthyphenmin\m@ne
4553 \bbl@hook@loadpatterns{#2}%
4554 % > luatex
4555 \ifnum\lefthyphenmin=\m@ne
4556 \else
4557 \expandafter\xdef\csname #1hyphenmins\endcsname{%
4558 \the\lefthyphenmin\the\righthyphenmin}%
4559 \fi
4560 \endgroup
4561 \def\bbl@tempa{#3}%
4562 \ifx\bbl@tempa\empty\else
4563 \bbl@hook@loadexceptions{#3}%
4564 % > luatex
4565 \fi
4566 \let\bbl@elt\relax
4567 \edef\bbl@languages{%
4568 \bbl@languages\bbl@elt{#1}\the\language}{#2}{\bbl@tempa}}%
4569 \ifnum\the\language=\z@
4570 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4571 \set@hyphenmins\tw@\thr@@\relax
4572 \else
4573 \expandafter\expandafter\expandafter\set@hyphenmins

```

```

4574     \csname #1hyphenmins\endcsname
4575   \fi
4576   \the\toks@
4577   \toks@{}%
4578 \fi}

```

### **\bbl@get@enc**

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

```

4579 \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.

```

4580 \def\bbl@hook@everylanguage#1{}
4581 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4582 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4583 \def\bbl@hook@loadkernel#1{%
4584   \def\addlanguage{\csname newlanguage\endcsname}%
4585   \def\adddialect##1##2{%
4586     \global\chardef##1##2\relax
4587     \wlog{\string##1 = a dialect from \string\language##2}}%
4588   \def\iflanguage##1{%
4589     \expandafter\ifx\csname l@##1\endcsname\relax
4590       \@nolanerr{##1}%
4591     \else
4592       \ifnum\csname l@##1\endcsname=\language
4593         \expandafter\expandafter\expandafter\@firstoftwo
4594       \else
4595         \expandafter\expandafter\expandafter\@secondoftwo
4596       \fi
4597     \fi}%
4598   \def\providehyphenmins##1##2{%
4599     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4600       \@namedef{##1hyphenmins}{##2}%
4601     \fi}%
4602   \def\set@hyphenmins##1##2{%
4603     \leftthyphenmin##1\relax
4604     \rightthyphenmin##2\relax}%
4605   \def\selectlanguage{%
4606     \errhelp{Selecting a language requires a package supporting it}%
4607     \errmessage{Not loaded}}%
4608   \let\foreignlanguage\selectlanguage
4609   \let\otherlanguage\selectlanguage
4610   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4611   \def\bbl@usehooks##1##2{% TODO. Temporary!!
4612   \def\setlocale{%
4613     \errhelp{Find an armchair, sit down and wait}%
4614     \errmessage{(babel) Not yet available}}%
4615   \let\uselocale\setlocale
4616   \let\locale\setlocale
4617   \let\selectlocale\setlocale
4618   \let\localename\setlocale
4619   \let\textlocale\setlocale
4620   \let\textlanguage\setlocale
4621   \let\languagetext\setlocale}
4622 \begingroup
4623   \def\AddBabelHook#1#2{%
4624     \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4625       \def\next{\toks1}%
4626     \else

```

```

4627     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4628     \fi
4629     \next}
4630 \ifx\directlua\undefined
4631     \ifx\XeTeXinputencoding\undefined\else
4632         \input xebabel.def
4633     \fi
4634 \else
4635     \input luababel.def
4636 \fi
4637 \openin1 = babel-\bbl@format.cfg
4638 \ifeof1
4639 \else
4640     \input babel-\bbl@format.cfg\relax
4641 \fi
4642 \closein1
4643 \endgroup
4644 \bbl@hook@loadkernel{switch.def}

```

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

```

4645 \openin1 = language.dat

```

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

```

4646 \def\language{english}%
4647 \ifeof1
4648     \message{I couldn't find the file language.dat,\space
4649             I will try the file hyphen.tex}
4650     \input hyphen.tex\relax
4651     \chardef\l@english\z@
4652 \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`.

```

4653     \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.

```

4654     \loop
4655         \endlinechar\m@ne
4656         \read1 to \bbl@line
4657         \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.

```

4658     \if T\ifeof1F\fi T\relax
4659     \ifx\bbl@line\empty\else
4660         \edef\bbl@line{\bbl@line\space\space\space}%
4661         \expandafter\process@line\bbl@line\relax
4662     \fi
4663 \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.

```

4664 \begingroup
4665     \def\bbl@elt#1#2#3#4{%
4666         \global\language=#2\relax
4667         \gdef\language{#1}%
4668         \def\bbl@elt##1##2##3##4{}}%

```

```

4669 \bbl@languages
4670 \endgroup
4671 \fi
4672 \closein1

```

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

```

4673 \if/\the\toks@/\else
4674 \errhelp{language.dat loads no language, only synonyms}
4675 \errmessage{Orphan language synonym}
4676 \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.

```

4677 \let\bbl@line\@undefined
4678 \let\process@line\@undefined
4679 \let\process@synonym\@undefined
4680 \let\process@language\@undefined
4681 \let\bbl@get@enc\@undefined
4682 \let\bbl@hyph@enc\@undefined
4683 \let\bbl@tempa\@undefined
4684 \let\bbl@hook@loadkernel\@undefined
4685 \let\bbl@hook@everylanguage\@undefined
4686 \let\bbl@hook@loadpatterns\@undefined
4687 \let\bbl@hook@loadexceptions\@undefined
4688 </patterns>

```

Here the code for `iniTeX` ends.

## 9 xetex + luatex: common stuff

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

```

4689 <<More package options>> ≡
4690 \chardef\bbl@bidimode\z@
4691 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4692 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4693 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4694 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4695 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4696 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4697 <</More package options>>

```

**\bblfont** 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`.

```

4698 <<Font selection>> ≡
4699 \bbl@trace{Font handling with fontspec}
4700 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4701 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfont}
4702 \DisableBabelHook{babel-fontspec}
4703 \@onlypreamble\bblfont
4704 \newcommand\bblfont[2][\% 1=langs/scripts 2=fam
4705 \bbl@foreach{#1}{\%
4706 \expandafter\ifx\csname date##1\endcsname\relax
4707 \IfFileExists{babel-##1.tex}{\%
4708 \babelprovide{##1}}{\%
4709 }{\%
4710 \fi}%
4711 \edef\bbl@tempa{#1}%
4712 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4713 \ifx\fontspec\@undefined

```

```

4714 \usepackage{fontspec}%
4715 \fi
4716 \EnableBabelHook{babel-fontspec}%
4717 \bbl@bblfont}
4718 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4719 \bbl@ifunset{\bbl@tempb family}%
4720 {\bbl@providefam{\bbl@tempb}}%
4721 {}%
4722 % For the default font, just in case:
4723 \bbl@ifunset{\bbl@lsys@\language\name}{\bbl@provide@lsys{\language\name}}{}%
4724 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4725 {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
4726 \bbl@exp{%
4727 \let<\bbl@tempb dflt@\language\name>\<\bbl@tempb dflt@>%
4728 \\\bbl@font@set<\bbl@tempb dflt@\language\name>%
4729 \<\bbl@tempb default>\<\bbl@tempb family>}}%
4730 {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4731 \bbl@csarg\def{\bbl@tempb dflt@##1}{<{#1}{#2}}}}}%

```

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

```

4732 \def\bbl@providefam#1{%
4733 \bbl@exp{%
4734 \\\newcommand\<#1default>{}% Just define it
4735 \\\bbl@add@list\\bbl@font@fams{#1}%
4736 \\\DeclareRobustCommand\<#1family>{%
4737 \\\not@math@alphabet\<#1family>\relax
4738 % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4739 \\\fontfamily\<#1default>%
4740 \<ifx>\\\UseHooks\\\@undefined\<else>\\\UseHook{#1family}\<fi>%
4741 \\\selectfont}%
4742 \\\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.

```

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

```



```

4773 \def\bbl@tempa{\bbl@nostdfont{}}% TODO. Don't use \bbl@tempa
4774 \bbl@foreach\bbl@font@fams{% don't gather with prev for
4775   \bbl@ifunset{\bbl@##1dflt@\language\language}%
4776   {\bbl@cs{famrst@##1}%
4777    \global\bbl@csarg\let{famrst@##1}\relax}%
4778   {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4779     \\bbl@add\\originalTeX{%
4780       \\bbl@font@rst{\bbl@cl{##1dflt}}%
4781       \<##1default>\<##1family>{##1}}%
4782     \\bbl@font@set{\bbl@##1dflt@\language\language}% the main part!
4783     \<##1default>\<##1family>}}}%
4784 \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`.

```

4785 \ifx\fbfamily\undefined\else % if latex
4786 \ifcase\bbl@engine % if pdftex
4787 \let\bbl@cckstdfont\relax
4788 \else
4789 \def\bbl@cckstdfont{%
4790   \begingroup
4791     \global\let\bbl@cckstdfont\relax
4792     \let\bbl@tempa\empty
4793     \bbl@foreach\bbl@font@fams{%
4794       \bbl@ifunset{\bbl@##1dflt@}%
4795       {\@nameuse{##1family}%
4796        \bbl@csarg\gdef{WFF@fbfamily}{}% Flag
4797        \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= \fbfamily\\}%
4798          \space\space\fontname\font\\}%
4799        \bbl@csarg\xdef{##1dflt@}{fbfamily}%
4800        \expandafter\xdef\csname ##1default\endcsname{fbfamily}%
4801        }%
4802     \ifx\bbl@tempa\empty\else
4803       \bbl@infowarn{The following font families will use the default\\%
4804         settings for all or some languages:\\%
4805         \bbl@tempa
4806         There is nothing intrinsically wrong with it, but\\%
4807         'babel' will no set Script and Language, which could\\%
4808         be relevant in some languages. If your document uses\\%
4809         these families, consider redefining them with \string\babelfont.\\%
4810         Reported}%
4811     \fi
4812   \endgroup}
4813 \fi
4814 \fi

```

Now the macros defining the font with `fontspec`.

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

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

```

4815 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4816   \bbl@xin@{<>}{#1}%
4817   \ifin@
4818     \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4819   \fi
4820   \bbl@exp{%
4821     \def\\#2#1{% eg, \rmdefault{\bbl@rmdflt@lang}
4822       \\bbl@ifsamestring{#2}{fbfamily}%

```

```

4823      {\#3%
4824      \\\bbl@ifsamestring{\f@series}{\bfdefault}{\bfseries}}}%
4825      \let\\bbl@tempa\relax}%
4826      {}}}
4827 %      TODO - next should be global?, but even local does its job. I'm
4828 %      still not sure -- must investigate:
4829 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4830 \let\bbl@tempe\bbl@mapselect
4831 \edef\bbl@tempb{\bbl@stripslash#4/}% Catcodes hack (better pass it).
4832 \bbl@exp{\\\bbl@replace\\bbl@tempb{\bbl@stripslash\family/}}}%
4833 \let\bbl@mapselect\relax
4834 \let\bbl@temp@fam#4%      eg, '\rmfamily', to be restored below
4835 \let#4\@empty %      Make sure \renewfontfamily is valid
4836 \bbl@exp{%
4837 \let\\bbl@temp@pfam<\bbl@stripslash#4\space> eg, '\rmfamily '
4838 <\keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4839 {\\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4840 <\keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4841 {\\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4842 \\\renewfontfamily\\#4%
4843 [\bbl@cl{lsys},% xetex removes unknown features :-(
4844 \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
4845 #2}}{#3}% ie \bbl@exp{..}{#3}
4846 \begingroup
4847 #4%
4848 \xdef#1{\f@family}%      eg, \bbl@rmdflt@lang{FreeSerif(0)}
4849 \endgroup % TODO. Find better tests:
4850 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4851 {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
4852 \ifin@
4853 \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
4854 \fi
4855 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4856 {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
4857 \ifin@
4858 \global\bbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
4859 \fi
4860 \let#4\bbl@temp@fam
4861 \bbl@exp{\let<\bbl@stripslash#4\space>}\bbl@temp@pfam
4862 \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.

```

4863 \def\bbl@font@rst#1#2#3#4{%
4864 \bbl@ccarg\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.

```

4865 \def\bbl@font@fams{rm,sf,tt}
4866 <</Font selection>>

```

## **\BabelFootnote** Footnotes

```

4867 <<{*Footnote changes}>> ≡
4868 \bbl@trace{Bidi footnotes}
4869 \ifnum\bbl@bidimode>\z@ % Any bidi=
4870 \def\bbl@footnote#1#2#3{%
4871 \@ifnextchar[%
4872 {\bbl@footnote@o{#1}{#2}{#3}}%
4873 {\bbl@footnote@x{#1}{#2}{#3}}}
4874 \long\def\bbl@footnote@x#1#2#3#4{%
4875 \bgroup
4876 \select@language@x{\bbl@main@language}%
4877 \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4878 \egroup}

```

```

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

```

## 10 Hooks for XeTeX and LuaTeX

### 10.1 XeTeX

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

Now, the code.

```

4914 (*xetex)
4915 \def\BabelStringsDefault{unicode}
4916 \let\xebbl@stop\relax
4917 \AddBabelHook{xetex}{encodedcommands}{%
4918   \def\bbl@tempa{#1}%
4919   \ifx\bbl@tempa\empty
4920     \XeTeXinputencoding"bytes"%
4921   \else
4922     \XeTeXinputencoding"#1"%
4923   \fi
4924   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4925 \AddBabelHook{xetex}{stopcommands}{%
4926   \xebbl@stop
4927   \let\xebbl@stop\relax}
4928 \def\bbl@input@classes{% Used in CJK intraspaces
4929   \input{load-unicode-xetex-classes.tex}%
4930   \let\bbl@input@classes\relax}
4931 \def\bbl@intraspace#1 #2 #3\@@{%
4932   \bbl@csarg\gdef{\xeisp@language}{%

```

```

4933 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4934 \def\bbl@intrapenalty#1\@@{%
4935 \bbl@csarg\gdef{xeipn@\language}%
4936 {\XeTeXlinebreakpenalty #1\relax}}
4937 \def\bbl@provide@intraspace{%
4938 \bbl@xin@{/s}{/\bbl@cl{lnbrk}}%
4939 \ifin@else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4940 \ifin@
4941 \bbl@ifunset{\bbl@intsp@\language}{}%
4942 {\expandafter\ifx\csname bbl@intsp@\language\endcsname\@empty\else
4943 \ifx\bbl@KVP@intraspace\@nnil
4944 \bbl@exp{%
4945 \\\bbl@intraspace\bbl@cl{intsp}\@@}%
4946 \fi
4947 \ifx\bbl@KVP@intrapenalty\@nnil
4948 \bbl@intrapenalty0\@@
4949 \fi
4950 \fi
4951 \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4952 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4953 \fi
4954 \ifx\bbl@KVP@intrapenalty\@nnil\else
4955 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4956 \fi
4957 \bbl@exp{%
4958 % TODO. Execute only once (but redundant):
4959 \\\bbl@add<extras\language>{%
4960 \XeTeXlinebreaklocale "\bbl@cl{tbc}"%
4961 <\bbl@xeisp@\language>%
4962 <\bbl@xeipn@\language>}%
4963 \\\bbl@tglobal<extras\language>%
4964 \\\bbl@add<noextras\language>{%
4965 \XeTeXlinebreaklocale ""}%
4966 \\\bbl@tglobal<noextras\language>}%
4967 \ifx\bbl@ispace\@undefined
4968 \gdef\bbl@ispace{\bbl@cl{xeisp}}%
4969 \ifx\AtBeginDocument\@notprerr
4970 \expandafter\@secondoftwo % to execute right now
4971 \fi
4972 \AtBeginDocument{\bbl@patchfont{\bbl@ispace}}%
4973 \fi}%
4974 \fi}
4975 \ifx\DisableBabelHook\@undefined\endinput\fi %%% TODO: why
4976 <@Font selection@>
4977 \def\bbl@provide@extra#1{}

```

## 11 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.

```

4978 \ifnum\Xe@alloc@intercharclass<\thr@@
4979 \Xe@alloc@intercharclass\thr@@
4980 \fi
4981 \chardef\bbl@xeiclass@default@=\z@
4982 \chardef\bbl@xeiclass@cjkideogram@=\@ne
4983 \chardef\bbl@xeiclass@cjkleftpunctuation@=\tw@
4984 \chardef\bbl@xeiclass@cjkrightpunctuation@=\thr@@
4985 \chardef\bbl@xeiclass@boundary@=4095
4986 \chardef\bbl@xeiclass@ignore@=4096

```

The machinery is activated with a hook (enabled only if actually used). Here \bbl@tempc is pre-set with \bbl@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 \bbl@upto, which is the previous char negated, as a flag to mark a range.

```

4987 \AddBabelHook{babel-interchar}{beforeextras}{%
4988   \nameuse{bbl@xechars@language}
4989 \DisableBabelHook{babel-interchar}
4990 \protected\def\bbl@charclass#1{%
4991   \ifnum\count@<\z@
4992     \count@-\count@
4993   \loop
4994     \bbl@exp{%
4995       \\babel@savevariable{\XeTeXcharclass`\Uchar\count@}%
4996       \XeTeXcharclass\count@ \bbl@tempc
4997       \ifnum\count@<`#1\relax
4998       \advance\count@@ne
4999     \repeat
5000 \else
5001   \babel@savevariable{\XeTeXcharclass`#1}%
5002   \XeTeXcharclass`#1 \bbl@tempc
5003 \fi
5004 \count@`#1\relax}

```

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

```

5005 \newcommand\bbl@ifinterchar[1]{%
5006   \let\bbl@tempa\@gobble % Assume to ignore
5007   \edef\bbl@tempb{\zap@space#1 \@empty}%
5008   \ifx\bbl@KVP@interchar\@nnil\else
5009     \bbl@replace\bbl@KVP@interchar{ }{,}%
5010     \bbl@foreach\bbl@tempb{%
5011       \bbl@xin{,##1,}{, \bbl@KVP@interchar,%
5012       \ifin@
5013       \let\bbl@tempa\@firstofone
5014       \fi}%
5015   \fi
5016   \bbl@tempa}
5017 \newcommand\IfBabelIntercharT[2]{%
5018   \bbl@carg\bbl@add{bbl@icsave@CurrentOption}{\bbl@ifinterchar{#1}{#2}}}%
5019 \newcommand\babelcharclass[3]{%
5020   \EnableBabelHook{babel-interchar}%
5021   \bbl@csarg\newXeTeXinterchartokenstate{xeclass@#2@#1}%
5022   \def\bbl@tempb##1{%
5023     \ifx##1\@empty\else
5024       \ifx##1-
5025         \bbl@upto
5026       \else
5027         \bbl@charclass{%
5028           \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
5029       \fi
5030       \expandafter\bbl@tempb
5031     \fi}%
5032   \bbl@ifunset{bbl@xechars@#1}%
5033   {\toks@{%
5034     \babel@savevariable\XeTeXinterchartokenstate
5035     \XeTeXinterchartokenstate\@ne
5036   }}%
5037   {\toks@\expandafter\expandafter\expandafter{%
5038     \csname bbl@xechars@#1\endcsname}}%
5039   \bbl@csarg\edef{xechars@#1}{%
5040     \the\toks@

```

```

5041 \bbl@usingxeclass\csname bbl@xeclass@#2@#1\endcsname
5042 \bbl@tempb#3\@empty}}
5043 \protected\def\bbl@usingxeclass#1{\count@ \z@ \let\bbl@tempc#1}
5044 \protected\def\bbl@upto{%
5045 \ifnum\count@>\z@
5046 \advance\count@\@ne
5047 \count@-\count@
5048 \else\ifnum\count@=\z@
5049 \bbl@charclass{-}%
5050 \else
5051 \bbl@error{double-hyphens-class}{\count@}{\count@}%
5052 \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 an intermediate macro, which can be 'disabled' with `\bbl@ic@<label>@<language>`.

```

5053 \def\bbl@ignoreinterchar{%
5054 \ifnum\language=\l@nohyphenation
5055 \expandafter\@gobble
5056 \else
5057 \expandafter\@firstofone
5058 \fi}
5059 \newcommand\babelinterchar[5][]{%
5060 \let\bbl@kv@label\@empty
5061 \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}%
5062 \@namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
5063 {\bbl@ignoreinterchar{#5}}%
5064 \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone
5065 \bbl@exp{\bbl@for{\bbl@tempa{\zap@space#3 \@empty}}}%
5066 \bbl@exp{\bbl@for{\bbl@tempb{\zap@space#4 \@empty}}}%
5067 \XeTeXinterchartoks
5068 \nameuse{bbl@xeclass@\bbl@tempa @#2}%
5069 \bbl@ifunset{bbl@xeclass@\bbl@tempa @#2}{\bbl@tempa @#2}%
5070 \nameuse{bbl@xeclass@\bbl@tempb @#2}%
5071 \bbl@ifunset{bbl@xeclass@\bbl@tempb @#2}{\bbl@tempb @#2}%
5072 = \expandafter{
5073 \csname bbl@ic@\bbl@kv@label @#2\expandafter\endcsname
5074 \csname\zap@space bbl@xeinter@\bbl@kv@label
5075 @#3@#4@#2 \@empty\endcsname}}}}
5076 \DeclareRobustCommand\enablelocaleinterchar[1]{%
5077 \bbl@ifunset{bbl@ic@#1@\language}%
5078 {\bbl@error{unknown-interchar}{#1}{\bbl@tempa @#2}}%
5079 {\bbl@csarg\let{ic@#1@\language}\@firstofone}}
5080 \DeclareRobustCommand\disablelocaleinterchar[1]{%
5081 \bbl@ifunset{bbl@ic@#1@\language}%
5082 {\bbl@error{unknown-interchar-b}{#1}{\bbl@tempa @#2}}%
5083 {\bbl@csarg\let{ic@#1@\language}\@gobble}}
5084 </xetex>

```

## 11.1 Layout

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

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

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

```

5085 < *xetex | texxet >
5086 \providecommand\bbl@provide@intraspace{}
5087 \bbl@trace{Redefinitions for bidi layout}
5088 \def\bbl@sspre@caption{% TODO: Unused!
5089 \bbl@exp{\everybox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
5090 \ifx\bbl@opt@layout\@nnil\else % if layout=..

```

```

5091 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
5092 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
5093 \ifnum\bbl@bidimode>\z@ % TODO: always?
5094 \def\@hangfrom#1{%
5095   \setbox\@tempboxa\hbox{\{#1\}}%
5096   \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
5097   \noindent\box\@tempboxa}
5098 \def\raggedright{%
5099   \let\\\@centercr
5100   \bbl@startskip\z@skip
5101   \@rightskip\@flushglue
5102   \bbl@endskip\@rightskip
5103   \parindent\z@
5104   \parfillskip\bbl@startskip}
5105 \def\raggedleft{%
5106   \let\\\@centercr
5107   \bbl@startskip\@flushglue
5108   \bbl@endskip\z@skip
5109   \parindent\z@
5110   \parfillskip\bbl@endskip}
5111 \fi
5112 \IfBabelLayout{lists}
5113 {\bbl@sreplace\list
5114   {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
5115   \def\bbl@listleftmargin{%
5116     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
5117   \ifcase\bbl@engine
5118     \def\labelenumii{}\theenumii{}% pdftex doesn't reverse ()
5119     \def\p@enumiii{\p@enumii}\theenumii{}%
5120   \fi
5121   \bbl@sreplace\@verbatim
5122     {\leftskip\@totalleftmargin}%
5123     {\bbl@startskip\textwidth
5124       \advance\bbl@startskip-\linewidth}%
5125   \bbl@sreplace\@verbatim
5126     {\rightskip\z@skip}%
5127     {\bbl@endskip\z@skip}}%
5128 {}
5129 \IfBabelLayout{contents}
5130 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
5131   \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
5132 {}
5133 \IfBabelLayout{columns}
5134 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outpuhbox}%
5135   \def\bbl@outpuhbox#1{%
5136     \hb@xt@\textwidth{%
5137       \hskip\columnwidth
5138       \hfil
5139       {\normalcolor\vrule \@width\columnseprule}%
5140       \hfil
5141       \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
5142       \hskip-\textwidth
5143       \hb@xt@\columnwidth{\box\@outputbox \hss}%
5144       \hskip\columnsep
5145       \hskip\columnwidth}}}%
5146 {}
5147 <@Footnote changes@>
5148 \IfBabelLayout{footnotes}%
5149 {\BabelFootnote\footnote\languagename{}{}}%
5150 \BabelFootnote\localfootnote\languagename{}{}}%
5151 \BabelFootnote\mainfootnote{}{}{}%
5152 {}

```

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.

```

5153 \IfBabelLayout{counters*}%
5154   {\bbl@add\bbl@opt@layout{.counters.}%
5155     \AddToHook{shipout/before}{%
5156       \let\bbl@tempa\babelsublr
5157       \let\babelsublr\@firstofone
5158       \let\bbl@save@thepage\thepage
5159       \protected@edef\thepage{\thepage}%
5160       \let\babelsublr\bbl@tempa}%
5161     \AddToHook{shipout/after}{%
5162       \let\thepage\bbl@save@thepage}}}%
5163 \IfBabelLayout{counters}%
5164   {\let\bbl@latinarabic=\@arabic
5165     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5166     \let\bbl@asciroman=\@roman
5167     \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5168     \let\bbl@asciiRoman=\@Roman
5169     \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}}%
5170 \fi % end if layout
5171 </xetex | texxt>

```

## 11.2 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).

```

5172 <*texxt>
5173 \def\bbl@provide@extra#1{%
5174   % == auto-select encoding ==
5175   \ifx\bbl@encoding@select\off\@empty\else
5176     \bbl@ifunset{\bbl@encoding@#1}%
5177     {\def\@elt##1{,##1,}%
5178       \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5179       \count@\z@
5180       \bbl@foreach\bbl@tempe{%
5181         \def\bbl@tempd{##1}% Save last declared
5182         \advance\count@\@ne}%
5183       \ifnum\count@>\@ne % (1)
5184         \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5185         \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
5186         \bbl@replace\bbl@tempa{ },}%
5187         \global\bbl@csarg\let{encoding@#1}\@empty
5188         \bbl@xin@{,\bbl@tempd,},{,\bbl@tempa,}%
5189         \ifin@else % if main encoding included in ini, do nothing
5190           \let\bbl@tempb\relax
5191           \bbl@foreach\bbl@tempa{%
5192             \ifx\bbl@tempb\relax
5193               \bbl@xin@{,##1,},{,\bbl@tempe,}%
5194               \ifin@\def\bbl@tempb{##1}\fi
5195             \fi}%
5196           \ifx\bbl@tempb\relax\else
5197             \bbl@exp{%
5198               \global\<\bbl@add>\<\bbl@preextras@#1>\<\bbl@encoding@#1>}%
5199             \gdef\<\bbl@encoding@#1>{%
5200               \\babel@save\\f@encoding
5201               \\bbl@add\\originalTeX{\\selectfont}%
5202               \\fontencoding{\bbl@tempb}%
5203               \\selectfont}}}%
5204           \fi
5205         \fi
5206       \fi}%
5207   {}%
5208 \fi}

```



### 11.3 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`).

```

5210 ⟨*luatex⟩
5211 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
5212 \bbl@trace{Read language.dat}
5213 \ifx\bbl@readstream\undefined
5214   \csname newread\endcsname\bbl@readstream
5215 \fi
5216 \begingroup
5217   \toks@{}
5218   \count@% % 0=start, 1=0th, 2=normal
5219   \def\bbl@process@line#1#2 #3 #4 {%
5220     \ifx=#1%
5221       \bbl@process@synonym{#2}%
5222     \else
5223       \bbl@process@language{#1#2}{#3}{#4}%
5224     \fi
5225     \ignorespaces}
5226   \def\bbl@manylang{%
5227     \ifnum\bbl@last>\@ne
5228       \bbl@info{Non-standard hyphenation setup}%
5229     \fi
5230     \let\bbl@manylang\relax}
5231   \def\bbl@process@language#1#2#3{%
5232     \ifcase\count@
5233       \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
5234     \or
5235       \count@\tw@
5236     \fi

```

```

5237 \ifnum\count@=\tw@
5238 \expandafter\addlanguage\csname l@#1\endcsname
5239 \language\allocationnumber
5240 \chardef\bbl@last\allocationnumber
5241 \bbl@manylang
5242 \let\bbl@elt\relax
5243 \xdef\bbl@languages{%
5244 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
5245 \fi
5246 \the\toks@
5247 \toks@{}}
5248 \def\bbl@process@synonym@aux#1#2{%
5249 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
5250 \let\bbl@elt\relax
5251 \xdef\bbl@languages{%
5252 \bbl@languages\bbl@elt{#1}{#2}{}}}%
5253 \def\bbl@process@synonym#1{%
5254 \ifcase\count@
5255 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
5256 \or
5257 \ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}}%
5258 \else
5259 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
5260 \fi}
5261 \ifx\bbl@languages\undefined % Just a (sensible?) guess
5262 \chardef\l@english\z@
5263 \chardef\l@USenglish\z@
5264 \chardef\bbl@last\z@
5265 \global\@namedef{bbl@hyphendata@0}{\hyphen.tex}}
5266 \gdef\bbl@languages{%
5267 \bbl@elt{english}{0}{\hyphen.tex}}%
5268 \bbl@elt{USenglish}{0}{}}
5269 \else
5270 \global\let\bbl@languages@format\bbl@languages
5271 \def\bbl@elt#1#2#3#4{% Remove all except language 0
5272 \ifnum#2>\z@\else
5273 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
5274 \fi}%
5275 \xdef\bbl@languages{\bbl@languages}%
5276 \fi
5277 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}} % Define flags
5278 \bbl@languages
5279 \openin\bbl@readstream=language.dat
5280 \ifeof\bbl@readstream
5281 \bbl@warning{I couldn't find language.dat. No additional\\%
5282 patterns loaded. Reported}%
5283 \else
5284 \loop
5285 \endlinechar\m@ne
5286 \read\bbl@readstream to \bbl@line
5287 \endlinechar\^^M
5288 \if T\ifeof\bbl@readstream F\fi T\relax
5289 \ifx\bbl@line\empty\else
5290 \edef\bbl@line{\bbl@line\space\space\space}%
5291 \expandafter\bbl@process@line\bbl@line\relax
5292 \fi
5293 \repeat
5294 \fi
5295 \closein\bbl@readstream
5296 \endgroup
5297 \bbl@trace{Macros for reading patterns files}
5298 \def\bbl@get@enc#1:#2:#3@@@{\def\bbl@hyph@enc{#2}}
5299 \ifx\babelcatcodetablenum\undefined

```

```

5300 \ifx\newcatcodetable\@undefined
5301 \def\babelcatcodetablenum{5211}
5302 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5303 \else
5304 \newcatcodetable\babelcatcodetablenum
5305 \newcatcodetable\bbl@pattcodes
5306 \fi
5307 \else
5308 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5309 \fi
5310 \def\bbl@luapatterns#1#2{%
5311 \bbl@get@enc#1:.\@@@
5312 \setbox\z@\hbox\bgroup
5313 \beginingroup
5314 \savecatcodetable\babelcatcodetablenum\relax
5315 \initcatcodetable\bbl@pattcodes\relax
5316 \catcodetable\bbl@pattcodes\relax
5317 \catcode\#=6 \catcode\$_=3 \catcode\&=4 \catcode\^=7
5318 \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\~=13
5319 \catcode\@=11 \catcode\^^I=10 \catcode\^^J=12
5320 \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
5321 \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
5322 \catcode\`=12 \catcode\`=12 \catcode\"=12
5323 \input #1\relax
5324 \catcodetable\babelcatcodetablenum\relax
5325 \endgroup
5326 \def\bbl@tempa{#2}%
5327 \ifx\bbl@tempa\@empty\else
5328 \input #2\relax
5329 \fi
5330 \egroup}%
5331 \def\bbl@patterns@lua#1{%
5332 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5333 \csname l@#1\endcsname
5334 \edef\bbl@tempa{#1}%
5335 \else
5336 \csname l@#1:\f@encoding\endcsname
5337 \edef\bbl@tempa{#1:\f@encoding}%
5338 \fi\relax
5339 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5340 \@ifundefined{bbl@hyphendata@the\language}%
5341 {\def\bbl@elt##1##2##3##4{%
5342 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
5343 \def\bbl@tempb{##3}%
5344 \ifx\bbl@tempb\@empty\else % if not a synonymous
5345 \def\bbl@tempc{##3}##4}%
5346 \fi
5347 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5348 \fi}%
5349 \bbl@languages
5350 \@ifundefined{bbl@hyphendata@the\language}%
5351 {\bbl@info{No hyphenation patterns were set for\%
5352 language '\bbl@tempa'. Reported}}%
5353 {\expandafter\expandafter\expandafter\bbl@luapatterns
5354 \csname bbl@hyphendata@the\language\endcsname}}}%
5355 \endinput\fi

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

5356 \ifx\DisableBabelHook\@undefined
5357 \AddBabelHook{luatex}{everylanguage}{%
5358 \def\process@language##1##2##3{%
5359 \def\process@line####1####2 ####3 ####4 {}}}
5360 \AddBabelHook{luatex}{loadpatterns}{%

```

```

5361 \input #1\relax
5362 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5363 {{#1}}}}
5364 \AddBabelHook{luatex}{loadexceptions}{%
5365 \input #1\relax
5366 \def\bbl@tempb##1##2{{##1}{##2}}}%
5367 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5368 {\expandafter\expandafter\expandafter\bbl@tempb
5369 \csname bbl@hyphendata@the\language\endcsname}}
5370 \endinput\fi

```

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

```

5371 \beginingroup % TODO - to a lua file
5372 \catcode`\%=12
5373 \catcode`\'=12
5374 \catcode`\%=12
5375 \catcode`\:=12
5376 \directlua{
5377 Babel = Babel or {}
5378 function Babel.lua_error(e, a)
5379 tex.print([[noexpand\csname bbl@error\endcsname{]] ..
5380 e .. '}' .. (a or '') .. '}]{}')
5381 end
5382 function Babel.bytes(line)
5383 return line:gsub(".",
5384 function (chr) return unicode.utf8.char(string.byte(chr)) end)
5385 end
5386 function Babel.begin_process_input()
5387 if luatexbase and luatexbase.add_to_callback then
5388 luatexbase.add_to_callback('process_input_buffer',
5389 Babel.bytes, 'Babel.bytes')
5390 else
5391 Babel.callback = callback.find('process_input_buffer')
5392 callback.register('process_input_buffer', Babel.bytes)
5393 end
5394 end
5395 function Babel.end_process_input ()
5396 if luatexbase and luatexbase.remove_from_callback then
5397 luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5398 else
5399 callback.register('process_input_buffer', Babel.callback)
5400 end
5401 end
5402 function Babel.addpatterns(pp, lg)
5403 local lg = lang.new(lg)
5404 local pats = lang.patterns(lg) or ''
5405 lang.clear_patterns(lg)
5406 for p in pp:gmatch('[^%s]+') do
5407 ss = ''
5408 for i in string.utfcharacters(p:gsub('%d', '')) do
5409 ss = ss .. '%d?' .. i
5410 end
5411 ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
5412 ss = ss:gsub('%.%%d%?$', '%%.')
5413 pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5414 if n == 0 then
5415 tex.sprint(
5416 [[\string\csname space bbl@info\endcsname{New pattern: ]]
5417 .. p .. [}}]])
5418 pats = pats .. ' ' .. p
5419 else
5420 tex.sprint(

```

```

5421         [[\string\csname\space bbl@info\endcsname{Renew pattern: }]
5422         .. p .. [[]]])
5423     end
5424 end
5425 lang.patterns(lg, pats)
5426 end
5427 Babel.characters = Babel.characters or {}
5428 Babel.ranges = Babel.ranges or {}
5429 function Babel.hlist_has_bidi(head)
5430     local has_bidi = false
5431     local ranges = Babel.ranges
5432     for item in node.traverse(head) do
5433         if item.id == node.id'glyph' then
5434             local itemchar = item.char
5435             local chardata = Babel.characters[itemchar]
5436             local dir = chardata and chardata.d or nil
5437             if not dir then
5438                 for nn, et in ipairs(ranges) do
5439                     if itemchar < et[1] then
5440                         break
5441                     elseif itemchar <= et[2] then
5442                         dir = et[3]
5443                         break
5444                     end
5445                 end
5446             end
5447             if dir and (dir == 'al' or dir == 'r') then
5448                 has_bidi = true
5449             end
5450         end
5451     end
5452     return has_bidi
5453 end
5454 function Babel.set_chrngs_b (script, chrng)
5455     if chrng == '' then return end
5456     texio.write('Replacing ' .. script .. ' script ranges')
5457     Babel.script_blocks[script] = {}
5458     for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5459         table.insert(
5460             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5461     end
5462 end
5463 function Babel.discard_sublr(str)
5464     if str:find( [[\string\indexentry]] ) and
5465        str:find( [[\string\babelsublr]] ) then
5466         str = str:gsub( [[\string\babelsublr%s*(%b{})]],
5467             function(m) return m:sub(2,-2) end )
5468     end
5469     return str
5470 end
5471 }
5472 \endgroup
5473 \ifx\newattribute\@undefined\else % Test for plain
5474     \newattribute\bbl@attr@locale
5475     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5476     \AddBabelHook{luatex}{beforeextras}{%
5477         \setattribute\bbl@attr@locale\localeid}
5478 \fi
5479 \def\BabelStringsDefault{unicode}
5480 \let\luabbbl@stop\relax
5481 \AddBabelHook{luatex}{encodedcommands}{%
5482     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5483     \ifx\bbl@tempa\bbl@tempb\else

```

```

5484 \directlua{Babel.begin_process_input()}%
5485 \def\luabbl@stop{%
5486 \directlua{Babel.end_process_input()}}%
5487 \fi}%
5488 \AddBabelHook{luatex}{stopcommands}{%
5489 \luabbl@stop
5490 \let\luabbl@stop\relax}
5491 \AddBabelHook{luatex}{patterns}{%
5492 \ifundefined{bbl@hyphendata@the\language}%
5493 {\def\bbl@elt##1##2##3##4{%
5494 \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5495 \def\bbl@tempb{##3}%
5496 \ifx\bbl@tempb\@empty\else % if not a synonymous
5497 \def\bbl@tempc{{##3}{##4}}%
5498 \fi
5499 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5500 \fi}%
5501 \bbl@languages
5502 \ifundefined{bbl@hyphendata@the\language}%
5503 {\bbl@info{No hyphenation patterns were set for\%
5504 language '#2'. Reported}}%
5505 {\expandafter\expandafter\expandafter\bbl@luapatterns
5506 \csname bbl@hyphendata@the\language\endcsname}}}%
5507 \ifundefined{bbl@patterns@}{}%
5508 \begingroup
5509 \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5510 \ifin@else
5511 \ifx\bbl@patterns@\@empty\else
5512 \directlua{ Babel.addpatterns(
5513 [[\bbl@patterns@]], \number\language) }%
5514 \fi
5515 \ifundefined{bbl@patterns@#1}%
5516 \@empty
5517 {\directlua{ Babel.addpatterns(
5518 [[\space\csname bbl@patterns@#1\endcsname]],
5519 \number\language) }}%
5520 \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5521 \fi
5522 \endgroup}%
5523 \bbl@exp{%
5524 \bbl@ifunset{bbl@prehc@language}{}%
5525 {\bbl@ifblank{\bbl@cs{prehc@language}}}%
5526 {\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.

```

5527 \@onlypreamble\babelpatterns
5528 \AtEndOfPackage{%
5529 \newcommand\babelpatterns[2][\@empty]{%
5530 \ifx\bbl@patterns@relax
5531 \let\bbl@patterns@\@empty
5532 \fi
5533 \ifx\bbl@pttnlist\@empty\else
5534 \bbl@warning{%
5535 You must not intermingle \string\selectlanguage\space and\%
5536 \string\babelpatterns\space or some patterns will not\%
5537 be taken into account. Reported}%
5538 \fi
5539 \ifx\@empty#1%
5540 \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5541 \else
5542 \edef\bbl@tempb{\zap@space#1 \@empty}%

```

```

5543 \bbl@for\bbl@tempa\bbl@tempb{%
5544 \bbl@fixname\bbl@tempa
5545 \bbl@iflanguage\bbl@tempa{%
5546 \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5547 \@ifundefined{bbl@patterns@\bbl@tempa}%
5548 \@empty
5549 {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5550 #2}}}%
5551 \fi}}

```

## 11.4 Southeast Asian scripts

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

Replace regular (ie, implicit) discretionaries by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionaries are not touched. See Unicode UAX 14.

```

5552 % TODO - to a lua file -- or a logical place
5553 \directlua{
5554   Babel = Babel or {}
5555   Babel.linebreaking = Babel.linebreaking or {}
5556   Babel.linebreaking.before = {}
5557   Babel.linebreaking.after = {}
5558   Babel.locale = {} % Free to use, indexed by \localeid
5559   function Babel.linebreaking.add_before(func, pos)
5560     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5561     if pos == nil then
5562       table.insert(Babel.linebreaking.before, func)
5563     else
5564       table.insert(Babel.linebreaking.before, pos, func)
5565     end
5566   end
5567   function Babel.linebreaking.add_after(func)
5568     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5569     table.insert(Babel.linebreaking.after, func)
5570   end
5571 }
5572 \def\bbl@intraspace#1 #2 #3\@@{%
5573 \directlua{
5574   Babel = Babel or {}
5575   Babel.intraspaces = Babel.intraspaces or {}
5576   Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5577     {b = #1, p = #2, m = #3}
5578   Babel.locale_props[\the\localeid].intraspace = %
5579     {b = #1, p = #2, m = #3}
5580 }}
5581 \def\bbl@intrapenalty#1\@@{%
5582 \directlua{
5583   Babel = Babel or {}
5584   Babel.intrapenalties = Babel.intrapenalties or {}
5585   Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5586   Babel.locale_props[\the\localeid].intrapenalty = #1
5587 }}
5588 \begingroup
5589 \catcode`\%=12
5590 \catcode`\&=14
5591 \catcode`\'=12
5592 \catcode`\~ =12
5593 \gdef\bbl@seaintraspace{&
5594 \let\bbl@seaintraspace\relax
5595 \directlua{
5596   Babel = Babel or {}
5597   Babel.sea_enabled = true
5598   Babel.sea_ranges = Babel.sea_ranges or {}

```

```

5599 function Babel.set_chranges (script, chrng)
5600     local c = 0
5601     for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5602         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5603         c = c + 1
5604     end
5605 end
5606 function Babel.sea_disc_to_space (head)
5607     local sea_ranges = Babel.sea_ranges
5608     local last_char = nil
5609     local quad = 655360      &% 10 pt = 655360 = 10 * 65536
5610     for item in node.traverse(head) do
5611         local i = item.id
5612         if i == node.id'glyph' then
5613             last_char = item
5614         elseif i == 7 and item.subtype == 3 and last_char
5615             and last_char.char > 0x0C99 then
5616             quad = font.getfont(last_char.font).size
5617             for lg, rg in pairs(sea_ranges) do
5618                 if last_char.char > rg[1] and last_char.char < rg[2] then
5619                     lg = lg:sub(1, 4) &% Remove trailing number of, eg, Cyril
5620                     local intraspace = Babel.intraspaces[lg]
5621                     local intrapenalty = Babel.intrapenalties[lg]
5622                     local n
5623                     if intrapenalty ~= 0 then
5624                         n = node.new(14, 0)      &% penalty
5625                         n.penalty = intrapenalty
5626                         node.insert_before(head, item, n)
5627                     end
5628                     n = node.new(12, 13)      &% (glue, spaceskip)
5629                     node.setglue(n, intraspace.b * quad,
5630                                 intraspace.p * quad,
5631                                 intraspace.m * quad)
5632                     node.insert_before(head, item, n)
5633                     node.remove(head, item)
5634                 end
5635             end
5636         end
5637     end
5638 end
5639 }&
5640 \bbl@luahyphenate}

```

## 11.5 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.

```

5641 \catcode`\%=14
5642 \gdef\bbl@cjk intraspace{%
5643     \let\bbl@cjk intraspace\relax
5644     \directlua{
5645         Babel = Babel or {}
5646         require('babel-data-cjk.lua')
5647         Babel.cjk_enabled = true
5648         function Babel.cjk_linebreak(head)
5649             local GLYPH = node.id'glyph'
5650             local last_char = nil
5651             local quad = 655360      % 10 pt = 655360 = 10 * 65536
5652             local last_class = nil

```



```

5653     local last_lang = nil
5654
5655     for item in node.traverse(head) do
5656         if item.id == GLYPH then
5657
5658             local lang = item.lang
5659
5660             local LOCALE = node.get_attribute(item,
5661                 Babel.attr_locale)
5662             local props = Babel.locale_props[LOCALE]
5663
5664             local class = Babel.cjk_class[item.char].c
5665
5666             if props.cjk_quotes and props.cjk_quotes[item.char] then
5667                 class = props.cjk_quotes[item.char]
5668             end
5669
5670             if class == 'cp' then class = 'cl' % ]] as CL
5671             elseif class == 'id' then class = 'I'
5672             elseif class == 'cj' then class = 'I' % loose
5673             end
5674
5675             local br = 0
5676             if class and last_class and Babel.cjk_breaks[last_class][class] then
5677                 br = Babel.cjk_breaks[last_class][class]
5678             end
5679
5680             if br == 1 and props.linebreak == 'c' and
5681                 lang ~= \the\l@nohyphenation\space and
5682                 last_lang ~= \the\l@nohyphenation then
5683                 local intrapenalty = props.intrapenalty
5684                 if intrapenalty ~= 0 then
5685                     local n = node.new(14, 0)      % penalty
5686                     n.penalty = intrapenalty
5687                     node.insert_before(head, item, n)
5688                 end
5689                 local intraspace = props.intraspace
5690                 local n = node.new(12, 13)          % (glue, spaceskip)
5691                 node.setglue(n, intraspace.b * quad,
5692                     intraspace.p * quad,
5693                     intraspace.m * quad)
5694                 node.insert_before(head, item, n)
5695             end
5696
5697             if font.getfont(item.font) then
5698                 quad = font.getfont(item.font).size
5699             end
5700             last_class = class
5701             last_lang = lang
5702             else % if penalty, glue or anything else
5703                 last_class = nil
5704             end
5705         end
5706         lang.hyphenate(head)
5707     end
5708 }%
5709 \bbl@luahyphenate}
5710 \gdef\bbl@luahyphenate{%
5711 \let\bbl@luahyphenate\relax
5712 \directlua{
5713     luatexbase.add_to_callback('hyphenate',
5714         function (head, tail)
5715             if Babel.linebreaking.before then

```

```

5716         for k, func in ipairs(Babel.linebreaking.before) do
5717             func(head)
5718         end
5719     end
5720     lang.hyphenate(head)
5721     if Babel.cjk_enabled then
5722         Babel.cjk_linebreak(head)
5723     end
5724     if Babel.linebreaking.after then
5725         for k, func in ipairs(Babel.linebreaking.after) do
5726             func(head)
5727         end
5728     end
5729     if Babel.sea_enabled then
5730         Babel.sea_disc_to_space(head)
5731     end
5732 end,
5733 'Babel.hyphenate')
5734 }
5735 }
5736 \endgroup
5737 \def\bbbl@provide@intraspace{%
5738   \bbbl@ifunset{\bbbl@intsp@\languagename}{}%
5739   {\expandafter\ifx\csname \bbbl@intsp@\languagename\endcsname\@empty\else
5740     \bbbl@xin@{/c}{\bbbl@cl{\lnbrk}}}%
5741     \ifin@           % cjk
5742     \bbbl@cjk@intraspace
5743     \directlua{
5744       Babel = Babel or {}
5745       Babel.locale_props = Babel.locale_props or {}
5746       Babel.locale_props[\the\localeid].linebreak = 'c'
5747     }%
5748     \bbbl@exp{\bbbl@intraspace\bbbl@cl{\intsp}\bbbl@cl{}%
5749     \ifx\bbbl@KVP@intrapenalty\@nnil
5750       \bbbl@intrapenalty0\@@
5751     \fi
5752   \else           % sea
5753     \bbbl@sea@intraspace
5754     \bbbl@exp{\bbbl@intraspace\bbbl@cl{\intsp}\bbbl@cl{}%
5755     \directlua{
5756       Babel = Babel or {}
5757       Babel.sea_ranges = Babel.sea_ranges or {}
5758       Babel.set_chranges('\bbbl@cl{\sbcp}',
5759       '\bbbl@cl{\chrng}')
5760     }%
5761     \ifx\bbbl@KVP@intrapenalty\@nnil
5762       \bbbl@intrapenalty0\@@
5763     \fi
5764   \fi
5765   \fi
5766   \ifx\bbbl@KVP@intrapenalty\@nnil\else
5767     \expandafter\bbbl@intrapenalty\bbbl@KVP@intrapenalty\@@
5768   \fi}}

```

## 11.6 Arabic justification

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

```

5769 \ifnum\bbbl@bidimode>100 \ifnum\bbbl@bidimode<200
5770 \def\bbblar@chars{%
5771   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5772   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5773   0640,0641,0642,0643,0644,0645,0646,0647,0649}

```

```

5774 \def\bblar@elongated{%
5775   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5776   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5777   0649,064A}
5778 \begingroup
5779   \catcode`\_:=11 \catcode`\:=11
5780   \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5781 \endgroup
5782 \gdef\bblar@arabicjust{% TODO. Allow for several locales.
5783   \let\bblar@arabicjust\relax
5784   \newattribute\bblar@kashida
5785   \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5786   \bblar@kashida=\z@
5787   \bblar@patchfont{\bblar@parsejalt}}%
5788   \directlua{
5789     Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5790     Babel.arabic.elong_map[\the\localeid] = {}
5791     luatexbase.add_to_callback('post_linebreak_filter',
5792       Babel.arabic.justify, 'Babel.arabic.justify')
5793     luatexbase.add_to_callback('hpack_filter',
5794       Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5795   }}%

```

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

```

5796 \def\bblar@fetchjalt#1#2#3#4{%
5797   \bblar@exp{\bblar@foreach{#1}}{%
5798     \bblar@ifunset\bblar@JE@##1}%
5799     {\setbox\z@\hbox{\textdir TRT ^^^200d\char"##1#2}}%
5800     {\setbox\z@\hbox{\textdir TRT ^^^200d\char"\@nameuse\bblar@JE@##1#2}}%
5801   \directlua{%
5802     local last = nil
5803     for item in node.traverse(tex.box[0].head) do
5804       if item.id == node.id'glyph' and item.char > 0x600 and
5805         not (item.char == 0x200D) then
5806         last = item
5807       end
5808     end
5809     Babel.arabic.#3['##1#4'] = last.char
5810   }}

```

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

```

5811 \gdef\bblar@parsejalt{%
5812   \ifx\addfontfeature\undefined\else
5813     \bblar@xin@{/e}{/\bblar@cl{\lnbrk}}%
5814     \ifin@
5815       \directlua{%
5816         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5817           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5818           tex.print([\string\csname\space bblar@parsejalti\endcsname])
5819         end
5820       }%
5821     \fi
5822   \fi}
5823 \gdef\bblar@parsejalti{%
5824   \begingroup
5825     \let\bblar@parsejalt\relax % To avoid infinite loop
5826     \edef\bblar@tempb{\fontid\font}%
5827     \bblar@nofswarn
5828     \bblar@fetchjalt\bblar@elongated{}{from}}%
5829     \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
5830     \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
5831     \addfontfeature{RawFeature+=jalt}%
5832     % \@namedef\bblar@JE@0643{06AA}% todo: catch medial kaf

```

```

5833 \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5834 \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
5835 \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
5836 \directlua{%
5837     for k, v in pairs(Babel.arabic.from) do
5838         if Babel.arabic.dest[k] and
5839             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5840             Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5841                 [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5842         end
5843     end
5844 }%
5845 \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```

5846 \begingroup
5847 \catcode`#=11
5848 \catcode`~ =11
5849 \directlua{
5850
5851 Babel.arabic = Babel.arabic or {}
5852 Babel.arabic.from = {}
5853 Babel.arabic.dest = {}
5854 Babel.arabic.justify_factor = 0.95
5855 Babel.arabic.justify_enabled = true
5856 Babel.arabic.kashida_limit = -1
5857
5858 function Babel.arabic.justify(head)
5859     if not Babel.arabic.justify_enabled then return head end
5860     for line in node.traverse_id(node.id'hlist', head) do
5861         Babel.arabic.justify_hlist(head, line)
5862     end
5863     return head
5864 end
5865
5866 function Babel.arabic.justify_hbox(head, gc, size, pack)
5867     local has_inf = false
5868     if Babel.arabic.justify_enabled and pack == 'exactly' then
5869         for n in node.traverse_id(12, head) do
5870             if n.stretch_order > 0 then has_inf = true end
5871         end
5872         if not has_inf then
5873             Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5874         end
5875     end
5876     return head
5877 end
5878
5879 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5880     local d, new
5881     local k_list, k_item, pos_inline
5882     local width, width_new, full, k_curr, wt_pos, goal, shift
5883     local subst_done = false
5884     local elong_map = Babel.arabic.elong_map
5885     local cnt
5886     local last_line
5887     local GLYPH = node.id'glyph'
5888     local KASHIDA = Babel.attr_kashida
5889     local LOCALE = Babel.attr_locale
5890
5891     if line == nil then
5892         line = {}
5893         line.glue_sign = 1

```

```

5894     line.glue_order = 0
5895     line.head = head
5896     line.shift = 0
5897     line.width = size
5898 end
5899
5900 % Exclude last line. todo. But-- it discards one-word lines, too!
5901 % ? Look for glue = 12:15
5902 if (line.glue_sign == 1 and line.glue_order == 0) then
5903     elongs = {}      % Stores elongated candidates of each line
5904     k_list = {}      % And all letters with kashida
5905     pos_inline = 0   % Not yet used
5906
5907     for n in node.traverse_id(GLYPH, line.head) do
5908         pos_inline = pos_inline + 1 % To find where it is. Not used.
5909
5910         % Elongated glyphs
5911         if elong_map then
5912             local locale = node.get_attribute(n, LOCALE)
5913             if elong_map[locale] and elong_map[locale][n.font] and
5914                 elong_map[locale][n.font][n.char] then
5915                 table.insert(elongs, {node = n, locale = locale} )
5916                 node.set_attribute(n.prev, KASHIDA, 0)
5917             end
5918         end
5919
5920         % Tatwil
5921         if Babel.kashida_wts then
5922             local k_wt = node.get_attribute(n, KASHIDA)
5923             if k_wt > 0 then % todo. parameter for multi inserts
5924                 table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5925             end
5926         end
5927
5928     end % of node.traverse_id
5929
5930     if #elongs == 0 and #k_list == 0 then goto next_line end
5931     full = line.width
5932     shift = line.shift
5933     goal = full * Babel.arabic.justify_factor % A bit crude
5934     width = node.dimensions(line.head)      % The 'natural' width
5935
5936     % == Elongated ==
5937     % Original idea taken from 'chickenize'
5938     while (#elongs > 0 and width < goal) do
5939         subst_done = true
5940         local x = #elongs
5941         local curr = elongs[x].node
5942         local oldchar = curr.char
5943         curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5944         width = node.dimensions(line.head) % Check if the line is too wide
5945         % Substitute back if the line would be too wide and break:
5946         if width > goal then
5947             curr.char = oldchar
5948             break
5949         end
5950         % If continue, pop the just substituted node from the list:
5951         table.remove(elongs, x)
5952     end
5953
5954     % == Tatwil ==
5955     if #k_list == 0 then goto next_line end
5956

```

```

5957 width = node.dimensions(line.head) % The 'natural' width
5958 k_curr = #k_list % Traverse backwards, from the end
5959 wt_pos = 1
5960
5961 while width < goal do
5962     subst_done = true
5963     k_item = k_list[k_curr].node
5964     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5965         d = node.copy(k_item)
5966         d.char = 0x0640
5967         d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
5968         d.xoffset = 0
5969         line.head, new = node.insert_after(line.head, k_item, d)
5970         width_new = node.dimensions(line.head)
5971         if width > goal or width == width_new then
5972             node.remove(line.head, new) % Better compute before
5973             break
5974         end
5975         if Babel.fix_diacr then
5976             Babel.fix_diacr(k_item.next)
5977         end
5978         width = width_new
5979     end
5980     if k_curr == 1 then
5981         k_curr = #k_list
5982         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5983     else
5984         k_curr = k_curr - 1
5985     end
5986 end
5987
5988 % Limit the number of tatweel by removing them. Not very efficient,
5989 % but it does the job in a quite predictable way.
5990 if Babel.arabic.kashida_limit > -1 then
5991     cnt = 0
5992     for n in node.traverse_id(GLYPH, line.head) do
5993         if n.char == 0x0640 then
5994             cnt = cnt + 1
5995             if cnt > Babel.arabic.kashida_limit then
5996                 node.remove(line.head, n)
5997             end
5998         else
5999             cnt = 0
6000         end
6001     end
6002 end
6003
6004 ::next_line::
6005
6006 % Must take into account marks and ins, see luatex manual.
6007 % Have to be executed only if there are changes. Investigate
6008 % what's going on exactly.
6009 if subst_done and not gc then
6010     d = node.hpack(line.head, full, 'exactly')
6011     d.shift = shift
6012     node.insert_before(head, line, d)
6013     node.remove(head, line)
6014 end
6015 end % if process line
6016 end
6017 }
6018 \endgroup
6019 \fi\fi % ends Arabic just block: \ifnum\bbl@bidimode>100...

```

## 11.7 Common stuff

6020 <@Font selection@>

## 11.8 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.

```
6021% TODO - to a lua file
6022 \directlua{
6023 Babel.script_blocks = {
6024   ['dflt'] = {},
6025   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
6026               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFD}, {0x1EE0, 0x1EEF}},
6027   ['Armn'] = {{0x0530, 0x058F}},
6028   ['Beng'] = {{0x0980, 0x09FF}},
6029   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
6030   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
6031   ['Cyril'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
6032               {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
6033   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
6034   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
6035               {0xAB00, 0xAB2F}},
6036   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
6037   % Don't follow strictly Unicode, which places some Coptic letters in
6038   % the 'Greek and Coptic' block
6039   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
6040   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
6041               {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
6042               {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
6043               {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
6044               {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
6045               {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
6046   ['Hebr'] = {{0x0590, 0x05FF}},
6047   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
6048               {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
6049   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
6050   ['Knda'] = {{0x0C80, 0x0CFF}},
6051   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
6052               {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
6053               {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
6054   ['Lao'] = {{0x0E80, 0x0EFF}},
6055   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
6056               {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
6057               {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
6058   ['Mahj'] = {{0x11150, 0x1117F}},
6059   ['Mlym'] = {{0x0D00, 0x0D7F}},
6060   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
6061   ['Orya'] = {{0x0B00, 0x0B7F}},
6062   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
6063   ['Syr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
6064   ['Taml'] = {{0x0B80, 0x0BFF}},
6065   ['Telu'] = {{0x0C00, 0x0C7F}},
6066   ['Tfng'] = {{0x2D30, 0x2D7F}},
6067   ['Thai'] = {{0x0E00, 0x0E7F}},
6068   ['Tibt'] = {{0x0F00, 0x0FFF}},
6069   ['Vaii'] = {{0xA500, 0xA63F}},
6070   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
```

```

6071 }
6072
6073 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
6074 Babel.script_blocks.Hant = Babel.script_blocks.Hans
6075 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
6076
6077 function Babel.locale_map(head)
6078   if not Babel.locale_mapped then return head end
6079
6080   local LOCALE = Babel.attr_locale
6081   local GLYPH = node.id('glyph')
6082   local inmath = false
6083   local toloc_save
6084   for item in node.traverse(head) do
6085     local toloc
6086     if not inmath and item.id == GLYPH then
6087       % Optimization: build a table with the chars found
6088       if Babel.chr_to_loc[item.char] then
6089         toloc = Babel.chr_to_loc[item.char]
6090       else
6091         for lc, maps in pairs(Babel.loc_to_scr) do
6092           for _, rg in pairs(maps) do
6093             if item.char >= rg[1] and item.char <= rg[2] then
6094               Babel.chr_to_loc[item.char] = lc
6095               toloc = lc
6096               break
6097             end
6098           end
6099         end
6100         % Treat composite chars in a different fashion, because they
6101         % 'inherit' the previous locale.
6102         if (item.char >= 0x0300 and item.char <= 0x036F) or
6103            (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
6104            (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
6105           Babel.chr_to_loc[item.char] = -2000
6106           toloc = -2000
6107         end
6108         if not toloc then
6109           Babel.chr_to_loc[item.char] = -1000
6110         end
6111       end
6112       if toloc == -2000 then
6113         toloc = toloc_save
6114       elseif toloc == -1000 then
6115         toloc = nil
6116       end
6117       if toloc and Babel.locale_props[toloc] and
6118          Babel.locale_props[toloc].letters and
6119          tex.getcatcode(item.char) \string~= 11 then
6120         toloc = nil
6121       end
6122       if toloc and Babel.locale_props[toloc].script
6123          and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6124          and Babel.locale_props[toloc].script ==
6125          Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6126         toloc = nil
6127       end
6128       if toloc then
6129         if Babel.locale_props[toloc].lg then
6130           item.lang = Babel.locale_props[toloc].lg
6131           node.set_attribute(item, LOCALE, toloc)
6132         end
6133         if Babel.locale_props[toloc]['/'..item.font] then

```



```

6134         item.font = Babel.locale_props[toloc]['/'..item.font]
6135     end
6136 end
6137 toloc_save = toloc
6138 elseif not inmath and item.id == 7 then % Apply recursively
6139     item.replace = item.replace and Babel.locale_map(item.replace)
6140     item.pre      = item.pre and Babel.locale_map(item.pre)
6141     item.post     = item.post and Babel.locale_map(item.post)
6142 elseif item.id == node.id'math' then
6143     inmath = (item.subtype == 0)
6144 end
6145 end
6146 return head
6147 end
6148 }

```

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

```

6149 \newcommand\babelcharproperty[1]{%
6150   \count@=#1\relax
6151   \ifvmode
6152     \expandafter\bbl@chprop
6153   \else
6154     \bbl@error{charproperty-only-vertical}{}{}{}%
6155   \fi}
6156 \newcommand\bbl@chprop[3][\the\count@]{%
6157   \@tempcnta=#1\relax
6158   \bbl@ifunset{bbl@chprop@#2}% {unknown-char-property}
6159   {\bbl@error{unknown-char-property}{}{#2}{}%
6160   }%
6161   \loop
6162     \bbl@cs{chprop@#2}{#3}%
6163   \ifnum\count@<\@tempcnta
6164     \advance\count@\@ne
6165   \repeat}
6166 \def\bbl@chprop@direction#1{%
6167   \directlua{
6168     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6169     Babel.characters[\the\count@]['d'] = '#1'
6170   }}
6171 \let\bbl@chprop@bc\bbl@chprop@direction
6172 \def\bbl@chprop@mirror#1{%
6173   \directlua{
6174     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6175     Babel.characters[\the\count@]['m'] = '\number#1'
6176   }}
6177 \let\bbl@chprop@bmg\bbl@chprop@mirror
6178 \def\bbl@chprop@linebreak#1{%
6179   \directlua{
6180     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6181     Babel.cjk_characters[\the\count@]['c'] = '#1'
6182   }}
6183 \let\bbl@chprop@lb\bbl@chprop@linebreak
6184 \def\bbl@chprop@locale#1{%
6185   \directlua{
6186     Babel.chr_to_loc = Babel.chr_to_loc or {}
6187     Babel.chr_to_loc[\the\count@] =
6188       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
6189   }}

```

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.

```

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

```

6192 }

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 `function(m) return m[1]..m[1]..'-' end`, where  $m$  are the matches returned after applying the pattern. With a mapped capture the functions are similar to `function(m) return Babel.capt_map(m[1],1) end`, where the last argument identifies the mapping to be applied to  $m[1]$ . The way it is carried out is somewhat tricky, but the effect is not dissimilar to `lua load` – save the code as string in a  $\TeX$  macro, and expand this macro at the appropriate place. As `\directlua` does not take into account the current catcode of `@`, we just avoid this character in macro names (which explains the internal group, too).

```

6193 \begingroup
6194 \catcode`\~ = 12
6195 \catcode`\% = 12
6196 \catcode`\& = 14
6197 \catcode`\| = 12
6198 \gdef\babelprehyphenation{%
6199   \ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}]
6200 \gdef\babelposthyphenation{%
6201   \ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}]
6202 \gdef\bbl@settransform#1[#2]#3#4#5{%
6203   \ifcase#1
6204     \bbl@activateprehyphen
6205   \or
6206     \bbl@activateposthyphen
6207   \fi
6208 \begingroup
6209   \def\babeltempa{\bbl@add@list\babeltempb}%
6210   \let\babeltempb@empty
6211   \def\bbl@tempa{#5}%
6212   \bbl@replace\bbl@tempa{,}{,}%
6213   \expandafter\bbl@foreach\expandafter{\bbl@tempa}{%
6214     \bbl@ifsamestring{##1}{remove}%
6215     {\bbl@add@list\babeltempb{nil}}%
6216     {\directlua{
6217       local rep = {[##1]=}
6218       rep = rep:gsub('^%s*(remove)%s$', 'remove = true')
6219       rep = rep:gsub('^%s*(insert)%s$', 'insert = true, ')
6220       rep = rep:gsub('^%s*(after)%s$', 'after = true, ')
6221       rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
6222       rep = rep:gsub('(node)%s*=%s*(%a+)%s*(%a*)', Babel.capture_node)
6223       rep = rep:gsub(&
6224         '(norule)%s*=%s*([%-d%.]+)%s+([%-d%.]+)%s+([%-d%.]+)',
6225         'norule = {' .. '%2, %3, %4' .. '})')
6226       if #1 == 0 or #1 == 2 then
6227         rep = rep:gsub(&
6228           '(space)%s*=%s*([%-d%.]+)%s+([%-d%.]+)%s+([%-d%.]+)',
6229           'space = {' .. '%2, %3, %4' .. '})')
6230         rep = rep:gsub(&
6231           '(spacefactor)%s*=%s*([%-d%.]+)%s+([%-d%.]+)%s+([%-d%.]+)',
6232           'spacefactor = {' .. '%2, %3, %4' .. '})')
6233         rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
6234       else
6235         rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
6236         rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
6237         rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
6238       end
6239       tex.print([[\string\babeltempa{[] .. rep .. []]])
6240     }]}%
6241 \bbl@foreach\babeltempb{%
6242   \bbl@forkv{##1}{%
6243     \in@{,###1,}{,nil,step,data,remove,insert,string,no,pre,no,&
6244     post,penalty,kashida,space,spacefactor,kern,node,after,norule,}%

```

```

6245     \ifin@else
6246     \bbl@error{bad-transform-option}{###1}{}}{}&%
6247     \fi}}&%
6248     \let\bbl@kv@attribute\relax
6249     \let\bbl@kv@label\relax
6250     \let\bbl@kv@fonts\@empty
6251     \bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}&%
6252     \ifx\bbl@kv@fonts\@empty\else\bbl@settransform\fi
6253     \ifx\bbl@kv@attribute\relax
6254     \ifx\bbl@kv@label\relax\else
6255     \bbl@exp{\bbl@trim@def\bbl@kv@fonts{\bbl@kv@fonts}}&%
6256     \bbl@replace\bbl@kv@fonts{ }{,}&%
6257     \edef\bbl@kv@attribute{\bbl@ATR@\bbl@kv@label @#3@\bbl@kv@fonts}&%
6258     \count@\z@
6259     \def\bbl@elt##1##2##3{&%
6260     \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
6261     {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
6262     {\count@\@ne}&%
6263     {\bbl@error{font-conflict-transforms}{}}{}}}&%
6264     }}&%
6265     \bbl@transform@list
6266     \ifnum\count@=\z@
6267     \bbl@exp{\global\bbl@add\bbl@transform@list
6268     {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
6269     \fi
6270     \bbl@ifunset{\bbl@kv@attribute}&%
6271     {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6272     {}&%
6273     \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
6274     \fi
6275     \else
6276     \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6277     \fi
6278     \directlua{
6279     local lbr = Babel.linebreaking.replacements[#1]
6280     local u = unicode.utf8
6281     local id, attr, label
6282     if #1 == 0 then
6283     id = \the\csname bbl@id@@#3\endcsname\space
6284     else
6285     id = \the\csname l@#3\endcsname\space
6286     end
6287     \ifx\bbl@kv@attribute\relax
6288     attr = -1
6289     \else
6290     attr = luatexbase.registernumber'\bbl@kv@attribute'
6291     \fi
6292     \ifx\bbl@kv@label\relax\else &% Same refs:
6293     label = [==[\bbl@kv@label]==]
6294     \fi
6295     &% Convert pattern:
6296     local patt = string.gsub([==[#4]==], '%s', '')
6297     if #1 == 0 then
6298     patt = string.gsub(patt, '|', ' ')
6299     end
6300     if not u.find(patt, '()', nil, true) then
6301     patt = '()' .. patt .. '()'
6302     end
6303     if #1 == 1 then
6304     patt = string.gsub(patt, '%(%)^', '^()')
6305     patt = string.gsub(patt, '%$(%)', '()$')
6306     end
6307     patt = u.gsub(patt, '{(.)}',

```

```

6308         function (n)
6309             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6310         end)
6311     patt = u.gsub(patt, '{(%x%x%x%x+)}',
6312         function (n)
6313             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6314         end)
6315     lbkr[id] = lbkr[id] or {}
6316     table.insert(lbkr[id],
6317         { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6318 }&%
6319 \endgroup}
6320 \endgroup
6321 \let\bbl@transfont@list\empty
6322 \def\bbl@settransfont{%
6323     \global\let\bbl@settransfont\relax % Execute only once
6324     \gdef\bbl@transfont{%
6325         \def\bbl@elt####1####2####3{%
6326             \bbl@ifblank{####3}%
6327                 {\count@tw@}% Do nothing if no fonts
6328                 {\count@z@
6329                     \bbl@vforeach{####3}{%
6330                         \def\bbl@tempd{#####1}%
6331                         \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6332                         \ifx\bbl@tempd\bbl@tempe
6333                             \count@one
6334                         \else\ifx\bbl@tempd\bbl@transfam
6335                             \count@one
6336                         \fi\fi}%
6337                     \ifcase\count@
6338                         \bbl@csarg\unsetattribute{ATR####2@####1@####3}%
6339                     \or
6340                         \bbl@csarg\setattribute{ATR####2@####1@####3}\@ne
6341                     \fi}}%
6342                 \bbl@transfont@list}%
6343     \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6344     \gdef\bbl@transfam{-unknown-}%
6345     \bbl@foreach\bbl@font@fams{%
6346         \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6347         \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6348         {\xdef\bbl@transfam{##1}}%
6349     }}}}
6350 \DeclareRobustCommand\enablelocaletransform[1]{%
6351     \bbl@ifunset{\bbl@ATR@#1@\language @}%
6352     {\bbl@error{transform-not-available}{#1}{}}}%
6353     {\bbl@csarg\setattribute{ATR@#1@\language @}\@ne}}
6354 \DeclareRobustCommand\disablelocaletransform[1]{%
6355     \bbl@ifunset{\bbl@ATR@#1@\language @}%
6356     {\bbl@error{transform-not-available-b}{#1}{}}}%
6357     {\bbl@csarg\unsetattribute{ATR@#1@\language @}}}%
6358 \def\bbl@activateposthyphen{%
6359     \let\bbl@activateposthyphen\relax
6360     \directlua{
6361         require('babel-transforms.lua')
6362         Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6363     }}
6364 \def\bbl@activateprehyphen{%
6365     \let\bbl@activateprehyphen\relax
6366     \directlua{
6367         require('babel-transforms.lua')
6368         Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6369     }}

```

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.

```
6370 \newcommand\localeprehyphenation[1]{%
6371 \directlua{ Babel.string_prehyphenation([=[#1]=], \the\localeid) }}
```

## 11.9 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 L<sup>A</sup>T<sub>E</sub>X. Just in case, consider the possibility it has not been loaded.

```
6372 \def\bbl@activate@preotf{%
6373 \let\bbl@activate@preotf\relax % only once
6374 \directlua{
6375   Babel = Babel or {}
6376   %
6377   function Babel.pre_otfload_v(head)
6378     if Babel.numbers and Babel.digits_mapped then
6379       head = Babel.numbers(head)
6380     end
6381     if Babel.bidi_enabled then
6382       head = Babel.bidi(head, false, dir)
6383     end
6384     return head
6385   end
6386   %
6387   function Babel.pre_otfload_h(head, gc, sz, pt, dir) %%% TODO
6388     if Babel.numbers and Babel.digits_mapped then
6389       head = Babel.numbers(head)
6390     end
6391     if Babel.bidi_enabled then
6392       head = Babel.bidi(head, false, dir)
6393     end
6394     return head
6395   end
6396   %
6397   luatexbase.add_to_callback('pre_linebreak_filter',
6398     Babel.pre_otfload_v,
6399     'Babel.pre_otfload_v',
6400     luatexbase.priority_in_callback('pre_linebreak_filter',
6401       'luaotfload.node_processor') or nil)
6402   %
6403   luatexbase.add_to_callback('hpack_filter',
6404     Babel.pre_otfload_h,
6405     'Babel.pre_otfload_h',
6406     luatexbase.priority_in_callback('hpack_filter',
6407       'luaotfload.node_processor') or nil)
6408 }
```

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.

```
6409 \breakafterdirmode=1
6410 \ifnum\bbl@bidimode>\@ne % Any bidi= except default (=1)
6411 \let\bbl@beforeforeign\leavevmode
6412 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6413 \RequirePackage{luatexbase}
6414 \bbl@activate@preotf
6415 \directlua{
6416   require('babel-data-bidi.lua')}
```

```

6417 \ifcase\expandafter\@gobbletwo\the\bbbl@bidimode\or
6418 require('babel-bidi-basic.lua')
6419 \or
6420 require('babel-bidi-basic-r.lua')
6421 table.insert(Babel.ranges, {0xE000, 0xF8FF, 'on'})
6422 table.insert(Babel.ranges, {0xF0000, 0xFFFFFD, 'on'})
6423 table.insert(Babel.ranges, {0x100000, 0x10FFFFD, 'on'})
6424 \fi}
6425 \newattribute\bbbl@attr@dir
6426 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbbl@attr@dir' }
6427 \bbbl@exp{\output{\bodydir\pagedir\the\output}}
6428 \fi
6429 \chardef\bbbl@thetextdir\z@
6430 \chardef\bbbl@thepardir\z@
6431 \def\bbbl@getluadir#1{%
6432 \directlua{
6433 if tex.#ldir == 'TLT' then
6434 tex.sprint('0')
6435 elseif tex.#ldir == 'TRT' then
6436 tex.sprint('1')
6437 end}}
6438 \def\bbbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6439 \ifcase#3\relax
6440 \ifcase\bbbl@getluadir{#1}\relax\else
6441 #2 TLT\relax
6442 \fi
6443 \else
6444 \ifcase\bbbl@getluadir{#1}\relax
6445 #2 TRT\relax
6446 \fi
6447 \fi}
6448 % ..00PPTT, with masks 0xC (par dir) and 0x3 (text dir)
6449 \def\bbbl@thedir{0}
6450 \def\bbbl@textdir#1{%
6451 \bbbl@setluadir{text}\textdir{#1}%
6452 \chardef\bbbl@thetextdir#1\relax
6453 \edef\bbbl@thedir{\the\numexpr\bbbl@thepardir*4+#1}%
6454 \setattribute\bbbl@attr@dir{\numexpr\bbbl@thepardir*4+#1}}
6455 \def\bbbl@pardir#1{% Used twice
6456 \bbbl@setluadir{par}\pardir{#1}%
6457 \chardef\bbbl@thepardir#1\relax}
6458 \def\bbbl@bodydir{\bbbl@setluadir{body}\bodydir}% Used once
6459 \def\bbbl@pagedir{\bbbl@setluadir{page}\pagedir}% Unused
6460 \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.

```

6461 \ifnum\bbbl@bidimode>\z@ % Any bidi=
6462 \def\bbbl@insidemath{0}%
6463 \def\bbbl@everymath{\def\bbbl@insidemath{1}}
6464 \def\bbbl@everydisplay{\def\bbbl@insidemath{2}}
6465 \frozen@everymath\expandafter{%
6466 \expandafter\bbbl@everymath\the\frozen@everymath}
6467 \frozen@everydisplay\expandafter{%
6468 \expandafter\bbbl@everydisplay\the\frozen@everydisplay}
6469 \AtBeginDocument{
6470 \directlua{
6471 function Babel.math_box_dir(head)
6472 if not (token.get_macro('bbbl@insidemath') == '0') then
6473 if Babel.hlist_has_bidi(head) then
6474 local d = node.new(node.id'dir')
6475 d.dir = '+TRT'
6476 node.insert_before(head, node.has_glyph(head), d)

```

```

6477         local inmath = false
6478         for item in node.traverse(head) do
6479             if item.id == 11 then
6480                 inmath = (item.subtype == 0)
6481             elseif not inmath then
6482                 node.set_attribute(item,
6483                     Babel.attr_dir, token.get_macro('bbl@thedir'))
6484             end
6485         end
6486     end
6487 end
6488 return head
6489 end
6490 luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6491     "Babel.math_box_dir", 0)
6492 if Babel.unset_atdir then
6493     luatexbase.add_to_callback("pre_linebreak_filter", Babel.unset_atdir,
6494         "Babel.unset_atdir")
6495     luatexbase.add_to_callback("hpack_filter", Babel.unset_atdir,
6496         "Babel.unset_atdir")
6497 end
6498 } }%
6499 \fi

```

Experimental. Tentative name.

```

6500 \DeclareRobustCommand\localebox[1]{%
6501     {\def\bbl@insidemath{0}%
6502         \mbox{\foreignlanguage{\language}{#1}}}}

```

## 11.10 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.

```

6503 \bbl@trace{Redefinitions for bidi layout}
6504 %
6505 << *More package options >> ≡
6506 \chardef\bbl@eqnpos\z@
6507 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6508 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6509 << /More package options >>
6510 %
6511 \ifnum\bbl@bidimode>\z@ % Any bidi=
6512     \matheqdirmode\@ne % A luatex primitive
6513     \let\bbl@eqnodir\relax
6514     \def\bbl@eqdel{()}
6515     \def\bbl@eqnum{%

```

```

6516 {\normalfont\normalcolor
6517 \expandafter\@firstoftwo\bbel@eqdel
6518 \theequation
6519 \expandafter\@secondoftwo\bbel@eqdel}}
6520 \def\bbel@puteqno#1{\eqno\hbox{#1}}
6521 \def\bbel@putleqno#1{\leqno\hbox{#1}}
6522 \def\bbel@eqno@flip#1{%
6523 \ifdim\predisplaysize=-\maxdimen
6524 \eqno
6525 \hb@xt@.01pt{%
6526 \hb@xt@\displaywidth{\hss{#1\glet\bbel@upset\@currentlabel}}\hss}%
6527 \else
6528 \leqno\hbox{#1\glet\bbel@upset\@currentlabel}%
6529 \fi
6530 \bbel@exp{\def\\@currentlabel{\[bbel@upset]}}
6531 \def\bbel@leqno@flip#1{%
6532 \ifdim\predisplaysize=-\maxdimen
6533 \leqno
6534 \hb@xt@.01pt{%
6535 \hss\hb@xt@\displaywidth{\#1\glet\bbel@upset\@currentlabel}\hss}%
6536 \else
6537 \eqno\hbox{#1\glet\bbel@upset\@currentlabel}%
6538 \fi
6539 \bbel@exp{\def\\@currentlabel{\[bbel@upset]}}
6540 \AtBeginDocument{%
6541 \ifx\bbel@noamsmath\relax\else
6542 \ifx\maketag@@@ \@undefined % Normal equation, eqnarray
6543 \AddToHook{env/equation/begin}{%
6544 \ifnum\bbel@thetextdir>z@
6545 \def\bbel@mathboxdir{\def\bbel@insidemath{1}}%
6546 \let\@eqnnum\bbel@eqnum
6547 \edef\bbel@eqnodir{\noexpand\bbel@textdir{\the\bbel@thetextdir}}%
6548 \chardef\bbel@thetextdirz@
6549 \bbel@add\normalfont{\bbel@eqnodir}%
6550 \ifcase\bbel@eqnpos
6551 \let\bbel@puteqno\bbel@eqno@flip
6552 \or
6553 \let\bbel@puteqno\bbel@leqno@flip
6554 \fi
6555 \fi}%
6556 \ifnum\bbel@eqnpos=\tw@ \else
6557 \def\endequation{\bbel@puteqno{\@eqnnum}$$\@ignoretrue}%
6558 \fi
6559 \AddToHook{env/eqnarray/begin}{%
6560 \ifnum\bbel@thetextdir>z@
6561 \def\bbel@mathboxdir{\def\bbel@insidemath{1}}%
6562 \edef\bbel@eqnodir{\noexpand\bbel@textdir{\the\bbel@thetextdir}}%
6563 \chardef\bbel@thetextdirz@
6564 \bbel@add\normalfont{\bbel@eqnodir}%
6565 \ifnum\bbel@eqnpos=\@ne
6566 \def\@eqnnum{%
6567 \setbox\z@\hbox{\bbel@eqnum}%
6568 \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6569 \else
6570 \let\@eqnnum\bbel@eqnum
6571 \fi
6572 \fi}
6573 % Hack. YA luatex bug?:
6574 \expandafter\bbel@sreplace\csname] \endcsname{${\eqno\kern.001pt$}%
6575 \else % amstex
6576 \bbel@exp{% Hack to hide maybe undefined conditionals:
6577 \chardef\bbel@eqnpos=0%
6578 \<iftagsleft@>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%

```



```

6579 \ifnum\bbled{eqnpos}=\@ne
6580 \let\bbled{ams@lap\hbox}
6581 \else
6582 \let\bbled{ams@lap\llap}
6583 \fi
6584 \ExplSyntaxOn % Required by \bbled{replace} with \intertext@
6585 \bbled{replace}\intertext@{\normalbaselines}%
6586 {\normalbaselines
6587 \ifx\bbled{eqnodir}\relax\else\bbled{pardir}\@ne\bbled{eqnodir}\fi}%
6588 \ExplSyntaxOff
6589 \def\bbled{ams@tagbox#1#2}{#1{\bbled{eqnodir#2}}}% #1=hbox|@lap|flip
6590 \ifx\bbled{ams@lap\hbox} % leqno
6591 \def\bbled{ams@flip#1}%
6592 \hbox to 0.01pt{\hss\hbox to\displaywidth{#{1}\hss}}}%
6593 \else % eqno
6594 \def\bbled{ams@flip#1}%
6595 \hbox to 0.01pt{\hbox to\displaywidth{\hss{#{1}}\hss}}}%
6596 \fi
6597 \def\bbled{ams@preset#1}%
6598 \def\bbled{mathboxdir}{\def\bbled{insidemath{1}}}%
6599 \ifnum\bbled{thetextdir}>\z@
6600 \edef\bbled{eqnodir}{\noexpand\bbled{textdir}{\the\bbled{thetextdir}}}%
6601 \bbled{replace}\textdef@{\hbox}{\bbled{ams@tagbox\hbox}}%
6602 \bbled{replace}\maketag@@@{\hbox}{\bbled{ams@tagbox#1}}%
6603 \fi}%
6604 \ifnum\bbled{eqnpos}=\tw@ \else
6605 \def\bbled{ams@equation}%
6606 \def\bbled{mathboxdir}{\def\bbled{insidemath{1}}}%
6607 \ifnum\bbled{thetextdir}>\z@
6608 \edef\bbled{eqnodir}{\noexpand\bbled{textdir}{\the\bbled{thetextdir}}}%
6609 \chardef\bbled{thetextdir}\z@
6610 \bbled{add}\normalfont{\bbled{eqnodir}}%
6611 \ifcase\bbled{eqnpos}
6612 \def\veqno##1##2{\bbled{eqno@flip}{##1##2}}%
6613 \or
6614 \def\veqno##1##2{\bbled{leqno@flip}{##1##2}}%
6615 \fi
6616 \fi}%
6617 \AddToHook{env/equation/begin}{\bbled{ams@equation}}%
6618 \AddToHook{env/equation*/begin}{\bbled{ams@equation}}%
6619 \fi
6620 \AddToHook{env/cases/begin}{\bbled{ams@preset}\bbled{ams@lap}}%
6621 \AddToHook{env/multline/begin}{\bbled{ams@preset}\hbox}%
6622 \AddToHook{env/gather/begin}{\bbled{ams@preset}\bbled{ams@lap}}%
6623 \AddToHook{env/gather*/begin}{\bbled{ams@preset}\bbled{ams@lap}}%
6624 \AddToHook{env/align/begin}{\bbled{ams@preset}\bbled{ams@lap}}%
6625 \AddToHook{env/align*/begin}{\bbled{ams@preset}\bbled{ams@lap}}%
6626 \AddToHook{env/alignat/begin}{\bbled{ams@preset}\bbled{ams@lap}}%
6627 \AddToHook{env/alignat*/begin}{\bbled{ams@preset}\bbled{ams@lap}}%
6628 \AddToHook{env/eqnalign/begin}{\bbled{ams@preset}\hbox}%
6629 % Hackish, for proper alignment. Don't ask me why it works!:
6630 \bbled{exp}% Avoid a 'visible' conditional
6631 \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\tag*{}>\<fi>}%
6632 \\\AddToHook{env/alignat*/end}{\<iftag@>\<else>\\tag*{}>\<fi>}}%
6633 \AddToHook{env/flalign/begin}{\bbled{ams@preset}\hbox}%
6634 \AddToHook{env/split/before}{%
6635 \def\bbled{mathboxdir}{\def\bbled{insidemath{1}}}%
6636 \ifnum\bbled{thetextdir}>\z@
6637 \bbled{ifsamestring}\@currentvir{equation}}%
6638 {\ifx\bbled{ams@lap\hbox} % leqno
6639 \def\bbled{ams@flip#1}%
6640 \hbox to 0.01pt{\hbox to\displaywidth{#{1}\hss}\hss}}%
6641 \else

```

```

6642             \def\bbl@ams@flip#1{%
6643                 \hbox to 0.01pt{\hss\hbox to \displaywidth{\hss{#1}}}%
6644             \fi}%
6645         {}%
6646     \fi}%
6647 \fi\fi}
6648 \fi
6649 \def\bbl@provide@extra#1{%
6650 % == Counters: mapdigits ==
6651 % Native digits
6652 \ifx\bbl@KVP@mapdigits\@nnil\else
6653     \bbl@ifunset{\bbl@dgnat\language\language}%
6654     {\RequirePackage{luatexbase}%
6655      \bbl@activate@preotf
6656      \directlua{
6657          Babel = Babel or {}  %% -> presets in luababel
6658          Babel.digits_mapped = true
6659          Babel.digits = Babel.digits or {}
6660          Babel.digits[\the\localeid] =
6661              table.pack(string.utfvalue('\bbl@cl{dgnat}'))
6662          if not Babel.numbers then
6663              function Babel.numbers(head)
6664                  local LOCALE = Babel.attr_locale
6665                  local GLYPH = node.id'glyph'
6666                  local inmath = false
6667                  for item in node.traverse(head) do
6668                      if not inmath and item.id == GLYPH then
6669                          local temp = node.get_attribute(item, LOCALE)
6670                          if Babel.digits[temp] then
6671                              local chr = item.char
6672                              if chr > 47 and chr < 58 then
6673                                  item.char = Babel.digits[temp][chr-47]
6674                              end
6675                          end
6676                      elseif item.id == node.id'math' then
6677                          inmath = (item.subtype == 0)
6678                      end
6679                  end
6680                  return head
6681              end
6682          end
6683      }%
6684  \fi
6685 % == transforms ==
6686 \ifx\bbl@KVP@transforms\@nnil\else
6687     \def\bbl@elt##1##2##3{%
6688         \in@{\$transforms.}{\$##1}%
6689         \ifin@
6690             \def\bbl@tempa{##1}%
6691             \bbl@replace\bbl@tempa{transforms.}{}%
6692             \bbl@carg\bbl@transforms{babel\bbl@tempa}{##2}{##3}%
6693         \fi}%
6694     \bbl@exp{%
6695         \\bbl@ifblank{\bbl@cl{dgnat}}%
6696         {\let\\bbl@tempa\relax}%
6697         {\def\\bbl@tempa{%
6698             \\bbl@elt{transforms.prehyphenation}%
6699             {digits.native.1.0}{([0-9])}%
6700             \\bbl@elt{transforms.prehyphenation}%
6701             {digits.native.1.1}{string={1\string|0123456789\string|\bbl@cl{dgnat}}}}}%
6702     \ifx\bbl@tempa\relax\else
6703         \toks@\expandafter\expandafter\expandafter{%
6704             \csname bbl@inidata@\language\endcsname}%

```

```

6705 \bbl@csarg\edef\inidata@\language\name}%
6706 \unexpanded\expandafter{\bbl@tempa}%
6707 \the\toks@}%
6708 \fi
6709 \csname bbl@inidata@\language\name\endcsname
6710 \bbl@release@transforms\relax % \relax closes the last item.
6711 \fi}

```

Start tabular here:

```

6712 \def\localerestoredirs{%
6713 \ifcase\bbl@thetextdir
6714 \ifnum\textdirection=\z@\else\textdir TLT\fi
6715 \else
6716 \ifnum\textdirection=\@ne\else\textdir TRT\fi
6717 \fi
6718 \ifcase\bbl@thepardir
6719 \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6720 \else
6721 \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6722 \fi}
6723 \IfBabelLayout{tabular}%
6724 {\chardef\bbl@tabular@mode\tw}% All RTL
6725 {\IfBabelLayout{notabular}%
6726 {\chardef\bbl@tabular@mode\z}%
6727 {\chardef\bbl@tabular@mode\@ne}}% Mixed, with LTR cols
6728 \ifnum\bbl@bidimode>\@ne % Any lua bidi= except default=1
6729 % Redefine: vrules mess up dirs. TODO: why?
6730 \def\@arstrut{\relax\copy\@arstrutbox}%
6731 \ifcase\bbl@tabular@mode\or % 1 = Mixed - default
6732 \let\bbl@parabefore\relax
6733 \AddToHook{para/before}{\bbl@parabefore}
6734 \AtBeginDocument{%
6735 \bbl@replace\@tabular{\$}{\$}%
6736 \def\bbl@insidemath{0}%
6737 \def\bbl@parabefore{\localerestoredirs}}%
6738 \ifnum\bbl@tabular@mode=\@ne
6739 \bbl@ifunset{\@tabclassz}{}%
6740 \bbl@exp{% Hide conditionals
6741 \\\bbl@sreplace\\ \@tabclassz
6742 {\<ifcase>\\ \@chnum}%
6743 {\localerestoredirs\<ifcase>\\ \@chnum}}}%
6744 \ifpackageloaded{colortbl}%
6745 {\bbl@sreplace\@classz
6746 {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6747 {\@ifpackageloaded{array}%
6748 {\bbl@exp{% Hide conditionals
6749 \\\bbl@sreplace\\ \@classz
6750 {\<ifcase>\\ \@chnum}%
6751 {\bgroup\\ \localerestoredirs\<ifcase>\\ \@chnum}%
6752 \\\bbl@sreplace\\ \@classz
6753 {\do@row@strut\<fi>}{\do@row@strut\<fi>\egroup}}}%
6754 {}}%
6755 \fi}%
6756 \or % 2 = All RTL - tabular
6757 \let\bbl@parabefore\relax
6758 \AddToHook{para/before}{\bbl@parabefore}%
6759 \AtBeginDocument{%
6760 \ifpackageloaded{colortbl}%
6761 {\bbl@replace\@tabular{\$}{\$}%
6762 \def\bbl@insidemath{0}%
6763 \def\bbl@parabefore{\localerestoredirs}}%
6764 \bbl@sreplace\@classz
6765 {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%

```

```

6766     {}}%
6767 \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.

```

6768 \AtBeginDocument{%
6769   \@ifpackageloaded{multicol}%
6770     {\toks@expandafter{\multi@column@out}%
6771      \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
6772   {}%
6773   \@ifpackageloaded{paracol}%
6774     {\edef\pcol@output{%
6775       \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
6776     {}}%
6777 \fi
6778 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout

```

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

```

6779 \ifnum\bbl@bidimode>\z@ % Any bidi=
6780   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6781     \bbl@exp{%
6782       \mathdir\the\bodydir
6783       #1%           Once entered in math, set boxes to restore values
6784       \def\\bbl@insidemath{0}%
6785       \<ifmmode>%
6786         \everyvbox{%
6787           \the\everyvbox
6788           \bodydir\the\bodydir
6789           \mathdir\the\mathdir
6790           \everyhbox{\the\everyhbox}%
6791           \everyvbox{\the\everyvbox}}%
6792         \everyhbox{%
6793           \the\everyhbox
6794           \bodydir\the\bodydir
6795           \mathdir\the\mathdir
6796           \everyhbox{\the\everyhbox}%
6797           \everyvbox{\the\everyvbox}}%
6798       \<fi>}}%
6799   \def\@hangfrom#1{%
6800     \setbox\@tempboxa\hbox{#1}%
6801     \hangindent\wd\@tempboxa
6802     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6803       \shapemode\@ne
6804     \fi
6805     \noindent\box\@tempboxa}
6806 \fi
6807 \IfBabelLayout{tabular}
6808   {\let\bbl@OL@tabular\@tabular
6809    \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6810    \let\bbl@NL@tabular\@tabular
6811    \AtBeginDocument{%
6812      \ifx\bbl@NL@tabular\@tabular\else
6813        \bbl@exp{\in{\bbl@nextfake}{\@tabular}}%
6814        \ifin\else
6815          \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6816        \fi
6817        \let\bbl@NL@tabular\@tabular
6818      \fi}}
6819   {}
6820 \IfBabelLayout{lists}

```

```

6821 {\let\bbL@OL@list\list
6822 \bbL@sreplace\list{\parshape}{\bbL@listparshape}%
6823 \let\bbL@NL@list\list
6824 \def\bbL@listparshape#1#2#3{%
6825 \parshape #1 #2 #3 %
6826 \ifnum\bbL@getluadir{page}=\bbL@getluadir{par}\else
6827 \shapemode\tw@
6828 \fi}}
6829 {}
6830 \IfBabelLayout{graphics}
6831 {\let\bbL@pictresetdir\relax
6832 \def\bbL@pictsetdir#1{%
6833 \ifcase\bbL@thetextdir
6834 \let\bbL@pictresetdir\relax
6835 \else
6836 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6837 \or\textdir TLT
6838 \else\bodydir TLT \textdir TLT
6839 \fi
6840 % \(\text|par)dir required in pgf:
6841 \def\bbL@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6842 \fi}%
6843 \AddToHook{env/picture/begin}{\bbL@pictsetdir\tw@}%
6844 \directlua{
6845 Babel.get_picture_dir = true
6846 Babel.picture_has_bidi = 0
6847 %
6848 function Babel.picture_dir (head)
6849 if not Babel.get_picture_dir then return head end
6850 if Babel.hlist_has_bidi(head) then
6851 Babel.picture_has_bidi = 1
6852 end
6853 return head
6854 end
6855 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6856 "Babel.picture_dir")
6857 }%
6858 \AtBeginDocument{%
6859 \def\LS@rot{%
6860 \setbox\@outputbox\vbox{%
6861 \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
6862 \long\def\put(#1,#2)#3{%
6863 \@killglue
6864 % Try:
6865 \ifx\bbL@pictresetdir\relax
6866 \def\bbL@tempc{0}%
6867 \else
6868 \directlua{
6869 Babel.get_picture_dir = true
6870 Babel.picture_has_bidi = 0
6871 }%
6872 \setbox\z@\hb@xt@\z@{%
6873 \@defaultunitsset\@tempdimc{#1}\unitlength
6874 \kern\@tempdimc
6875 #3\hss}% TODO: #3 executed twice (below). That's bad.
6876 \edef\bbL@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6877 \fi
6878 % Do:
6879 \@defaultunitsset\@tempdimc{#2}\unitlength
6880 \raise\@tempdimc\hb@xt@\z@{%
6881 \@defaultunitsset\@tempdimc{#1}\unitlength
6882 \kern\@tempdimc
6883 {\ifnum\bbL@tempc>\z@\bbL@pictresetdir\fi#3}\hss}%

```

```

6884 \ignorespaces}%
6885 \MakeRobust\put}%
6886 \AtBeginDocument
6887 {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir\@gobble}%
6888 \ifx\pgfpicture\undefined\else % TODO. Allow deactivate?
6889 \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6890 \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6891 \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6892 \fi
6893 \ifx\tikzpicture\undefined\else
6894 \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\tw@}%
6895 \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6896 \bbl@sreplace\tikz{\beginpgroup}{\beginpgroup\bbl@pictsetdir\tw@}%
6897 \fi
6898 \ifx\tcolorbox\undefined\else
6899 \def\tcb@drawing@env@begin{%
6900 \csname tcb@before@tcb@split@state\endcsname
6901 \bbl@pictsetdir\tw@
6902 \begin{\kvtcb@graphenv}%
6903 \tcb@bbdraw
6904 \tcb@apply@graph@patches}%
6905 \def\tcb@drawing@env@end{%
6906 \end{\kvtcb@graphenv}%
6907 \bbl@pictresetdir
6908 \csname tcb@after@tcb@split@state\endcsname}%
6909 \fi
6910 }}
6911 {}

```

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.

```

6912 \IfBabelLayout{counters*}%
6913 {\bbl@add\bbl@opt@layout{.counters.}%
6914 \directlua{
6915 \lua{
6916 \lua{
6917 }}}}
6918 \IfBabelLayout{counters}%
6919 {\let\bbl@OL@@@textsuperscript\textsuperscript
6920 \bbl@sreplace\textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6921 \let\bbl@latinarabic=\@arabic
6922 \let\bbl@OL@@@arabic\@arabic
6923 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
6924 \@ifpackagewith{babel}{bidi=default}%
6925 {\let\bbl@asciroman=\@roman
6926 \let\bbl@OL@@@roman\@roman
6927 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
6928 \let\bbl@asciiRoman=\@Roman
6929 \let\bbl@OL@@@roman\@Roman
6930 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6931 \let\bbl@OL@labelenumii\labelenumii
6932 \def\labelenumii{\theenumii}%
6933 \let\bbl@OL@p@enumiii\p@enumiii
6934 \def\p@enumiii{\p@enumii}\theenumii}}{}
6935 <@Footnote changes@>
6936 \IfBabelLayout{footnotes}%
6937 {\let\bbl@OL@footnote\footnote
6938 \BabelFootnote\footnote\languagename{}{}%
6939 \BabelFootnote\localfootnote\languagename{}{}%
6940 \BabelFootnote\mainfootnote{}{}{}%
6941 {}

```

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.

```

6942 \IfBabelLayout{extras}%
6943 {\bbl@ncarg\let\bbl@OL@underline{underline }%
6944 \bbl@carg\bbl@sreplace{underline }%
6945 {\$@@underline}{\bgroup\bbl@nextfake$@@underline}%
6946 \bbl@carg\bbl@sreplace{underline }%
6947 {\m@th$}{\m@th$\egroup}%
6948 \let\bbl@OL@LaTeXe\LaTeXe
6949 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6950 \if b\expandafter\@car\@f@series\@nil\boldmath\fi
6951 \babelsublr{%
6952 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
6953 {}
6954 \end{luatex}

```

## 11.11 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.

```

6955 (*transforms)
6956 Babel.linebreaking.replacements = {}
6957 Babel.linebreaking.replacements[0] = {} -- pre
6958 Babel.linebreaking.replacements[1] = {} -- post
6959
6960 function Babel.tovalue(v)
6961   if type(v) == 'string' then
6962     return loadstring('return ' .. v)()
6963   else
6964     return v
6965   end
6966 end
6967
6968 -- Discretionaries contain strings as nodes
6969 function Babel.str_to_nodes(fn, matches, base)
6970   local n, head, last
6971   if fn == nil then return nil end
6972   for s in string.utfvalues(fn(matches)) do
6973     if base.id == 7 then
6974       base = base.replace
6975     end
6976     n = node.copy(base)
6977     n.char = s
6978     if not head then
6979       head = n
6980     else
6981       last.next = n
6982     end
6983     last = n
6984   end
6985   return head
6986 end
6987
6988 Babel.fetch_subtext = {}

```

```

6989
6990 Babel.ignore_pre_char = function(node)
6991   return (node.lang == Babel.nohyphenation)
6992 end
6993
6994 -- Merging both functions doesn't seem feasible, because there are too
6995 -- many differences.
6996 Babel.fetch_subtext[0] = function(head)
6997   local word_string = ''
6998   local word_nodes = {}
6999   local lang
7000   local item = head
7001   local inmath = false
7002
7003   while item do
7004
7005     if item.id == 11 then
7006       inmath = (item.subtype == 0)
7007     end
7008
7009     if inmath then
7010       -- pass
7011
7012     elseif item.id == 29 then
7013       local locale = node.get_attribute(item, Babel.attr_locale)
7014
7015       if lang == locale or lang == nil then
7016         lang = lang or locale
7017         if Babel.ignore_pre_char(item) then
7018           word_string = word_string .. Babel.us_char
7019         else
7020           word_string = word_string .. unicode.utf8.char(item.char)
7021         end
7022         word_nodes[#word_nodes+1] = item
7023       else
7024         break
7025       end
7026
7027     elseif item.id == 12 and item.subtype == 13 then
7028       word_string = word_string .. ' '
7029       word_nodes[#word_nodes+1] = item
7030
7031     -- Ignore leading unrecognized nodes, too.
7032     elseif word_string ~= '' then
7033       word_string = word_string .. Babel.us_char
7034       word_nodes[#word_nodes+1] = item -- Will be ignored
7035     end
7036
7037     item = item.next
7038   end
7039
7040   -- Here and above we remove some trailing chars but not the
7041   -- corresponding nodes. But they aren't accessed.
7042   if word_string:sub(-1) == ' ' then
7043     word_string = word_string:sub(1,-2)
7044   end
7045   word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7046   return word_string, word_nodes, item, lang
7047 end
7048
7049 Babel.fetch_subtext[1] = function(head)
7050   local word_string = ''
7051   local word_nodes = {}

```



```

7052 local lang
7053 local item = head
7054 local inmath = false
7055
7056 while item do
7057
7058     if item.id == 11 then
7059         inmath = (item.subtype == 0)
7060     end
7061
7062     if inmath then
7063         -- pass
7064
7065     elseif item.id == 29 then
7066         if item.lang == lang or lang == nil then
7067             if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
7068                 lang = lang or item.lang
7069                 word_string = word_string .. unicode.utf8.char(item.char)
7070                 word_nodes[#word_nodes+1] = item
7071             end
7072         else
7073             break
7074         end
7075
7076     elseif item.id == 7 and item.subtype == 2 then
7077         word_string = word_string .. '='
7078         word_nodes[#word_nodes+1] = item
7079
7080     elseif item.id == 7 and item.subtype == 3 then
7081         word_string = word_string .. '|'
7082         word_nodes[#word_nodes+1] = item
7083
7084     -- (1) Go to next word if nothing was found, and (2) implicitly
7085     -- remove leading USs.
7086     elseif word_string == '' then
7087         -- pass
7088
7089     -- This is the responsible for splitting by words.
7090     elseif (item.id == 12 and item.subtype == 13) then
7091         break
7092
7093     else
7094         word_string = word_string .. Babel.us_char
7095         word_nodes[#word_nodes+1] = item -- Will be ignored
7096     end
7097
7098     item = item.next
7099 end
7100
7101 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7102 return word_string, word_nodes, item, lang
7103 end
7104
7105 function Babel.pre_hyphenate_replace(head)
7106     Babel.hyphenate_replace(head, 0)
7107 end
7108
7109 function Babel.post_hyphenate_replace(head)
7110     Babel.hyphenate_replace(head, 1)
7111 end
7112
7113 Babel.us_char = string.char(31)
7114

```

```

7115 function Babel.hyphenate_replace(head, mode)
7116   local u = unicode.utf8
7117   local lbkr = Babel.linebreaking.replacements[mode]
7118
7119   local word_head = head
7120
7121   while true do -- for each subtext block
7122
7123     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
7124
7125     if Babel.debug then
7126       print()
7127       print((mode == 0) and '@@@<' or '@@@>', w)
7128     end
7129
7130     if nw == nil and w == '' then break end
7131
7132     if not lang then goto next end
7133     if not lbkr[lang] then goto next end
7134
7135     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
7136     -- loops are nested.
7137     for k=1, #lbkr[lang] do
7138       local p = lbkr[lang][k].pattern
7139       local r = lbkr[lang][k].replace
7140       local attr = lbkr[lang][k].attr or -1
7141
7142       if Babel.debug then
7143         print('*****', p, mode)
7144       end
7145
7146       -- This variable is set in some cases below to the first *byte*
7147       -- after the match, either as found by u.match (faster) or the
7148       -- computed position based on sc if w has changed.
7149       local last_match = 0
7150       local step = 0
7151
7152       -- For every match.
7153       while true do
7154         if Babel.debug then
7155           print('====')
7156         end
7157         local new -- used when inserting and removing nodes
7158         local dummy_node -- used by after
7159
7160         local matches = { u.match(w, p, last_match) }
7161
7162         if #matches < 2 then break end
7163
7164         -- Get and remove empty captures (with ())'s, which return a
7165         -- number with the position), and keep actual captures
7166         -- (from (...)), if any, in matches.
7167         local first = table.remove(matches, 1)
7168         local last = table.remove(matches, #matches)
7169         -- Non re-fetched substrings may contain \31, which separates
7170         -- subsubstrings.
7171         if string.find(w:sub(first, last-1), Babel.us_char) then break end
7172
7173         local save_last = last -- with A()BC()D, points to D
7174
7175         -- Fix offsets, from bytes to unicode. Explained above.
7176         first = u.len(w:sub(1, first-1)) + 1
7177         last = u.len(w:sub(1, last-1)) -- now last points to C

```

```

7178
7179 -- This loop stores in a small table the nodes
7180 -- corresponding to the pattern. Used by 'data' to provide a
7181 -- predictable behavior with 'insert' (w_nodes is modified on
7182 -- the fly), and also access to 'remove'd nodes.
7183 local sc = first-1 -- Used below, too
7184 local data_nodes = {}
7185
7186 local enabled = true
7187 for q = 1, last-first+1 do
7188     data_nodes[q] = w_nodes[sc+q]
7189     if enabled
7190         and attr > -1
7191         and not node.has_attribute(data_nodes[q], attr)
7192     then
7193         enabled = false
7194     end
7195 end
7196
7197 -- This loop traverses the matched substring and takes the
7198 -- corresponding action stored in the replacement list.
7199 -- sc = the position in substr nodes / string
7200 -- rc = the replacement table index
7201 local rc = 0
7202
7203 ----- TODO. dummy_node?
7204 while rc < last-first+1 or dummy_node do -- for each replacement
7205     if Babel.debug then
7206         print('.....', rc + 1)
7207     end
7208     sc = sc + 1
7209     rc = rc + 1
7210
7211     if Babel.debug then
7212         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7213         local ss = ''
7214         for itt in node.traverse(head) do
7215             if itt.id == 29 then
7216                 ss = ss .. unicode.utf8.char(itt.char)
7217             else
7218                 ss = ss .. '{' .. itt.id .. '}'
7219             end
7220         end
7221         print('*****', ss)
7222     end
7223
7224     local crep = r[rc]
7225     local item = w_nodes[sc]
7226     local item_base = item
7227     local placeholder = Babel.us_char
7228     local d
7229
7230
7231     if crep and crep.data then
7232         item_base = data_nodes[crep.data]
7233     end
7234
7235     if crep then
7236         step = crep.step or step
7237     end
7238
7239     if crep and crep.after then
7240         crep.insert = true

```

```

7241         if dummy_node then
7242             item = dummy_node
7243         else -- TODO. if there is a node after?
7244             d = node.copy(item_base)
7245             head, item = node.insert_after(head, item, d)
7246             dummy_node = item
7247         end
7248     end
7249
7250     if crep and not crep.after and dummy_node then
7251         node.remove(head, dummy_node)
7252         dummy_node = nil
7253     end
7254
7255     if (not enabled) or (crep and next(crep) == nil) then -- = {}
7256         if step == 0 then
7257             last_match = save_last -- Optimization
7258         else
7259             last_match = utf8.offset(w, sc+step)
7260         end
7261         goto next
7262
7263     elseif crep == nil or crep.remove then
7264         node.remove(head, item)
7265         table.remove(w_nodes, sc)
7266         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7267         sc = sc - 1 -- Nothing has been inserted.
7268         last_match = utf8.offset(w, sc+1+step)
7269         goto next
7270
7271     elseif crep and crep.kashida then -- Experimental
7272         node.set_attribute(item,
7273             Babel.attr_kashida,
7274             crep.kashida)
7275         last_match = utf8.offset(w, sc+1+step)
7276         goto next
7277
7278     elseif crep and crep.string then
7279         local str = crep.string(matches)
7280         if str == '' then -- Gather with nil
7281             node.remove(head, item)
7282             table.remove(w_nodes, sc)
7283             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7284             sc = sc - 1 -- Nothing has been inserted.
7285         else
7286             local loop_first = true
7287             for s in string.utfvalues(str) do
7288                 d = node.copy(item_base)
7289                 d.char = s
7290                 if loop_first then
7291                     loop_first = false
7292                     head, new = node.insert_before(head, item, d)
7293                     if sc == 1 then
7294                         word_head = head
7295                     end
7296                     w_nodes[sc] = d
7297                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
7298                 else
7299                     sc = sc + 1
7300                     head, new = node.insert_before(head, item, d)
7301                     table.insert(w_nodes, sc, new)
7302                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7303                 end

```

```

7304         if Babel.debug then
7305             print('.....', 'str')
7306             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7307         end
7308     end -- for
7309     node.remove(head, item)
7310 end -- if ''
7311 last_match = utf8.offset(w, sc+1+step)
7312 goto next
7313
7314 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7315     d = node.new(7, 3) -- (disc, regular)
7316     d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7317     d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7318     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7319     d.attr = item_base.attr
7320     if crep.pre == nil then -- TeXbook p96
7321         d.penalty = crep.penalty or tex.hyphenpenalty
7322     else
7323         d.penalty = crep.penalty or tex.exhyphenpenalty
7324     end
7325     placeholder = '|'
7326     head, new = node.insert_before(head, item, d)
7327
7328 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7329     -- ERROR
7330
7331 elseif crep and crep.penalty then
7332     d = node.new(14, 0) -- (penalty, userpenalty)
7333     d.attr = item_base.attr
7334     d.penalty = crep.penalty
7335     head, new = node.insert_before(head, item, d)
7336
7337 elseif crep and crep.space then
7338     -- 655360 = 10 pt = 10 * 65536 sp
7339     d = node.new(12, 13) -- (glue, spaceskip)
7340     local quad = font.getfont(item_base.font).size or 655360
7341     node.setglue(d, crep.space[1] * quad,
7342                 crep.space[2] * quad,
7343                 crep.space[3] * quad)
7344     if mode == 0 then
7345         placeholder = ' '
7346     end
7347     head, new = node.insert_before(head, item, d)
7348
7349 elseif crep and crep.norule then
7350     -- 655360 = 10 pt = 10 * 65536 sp
7351     d = node.new(2, 3) -- (rule, empty) = \no*rule
7352     local quad = font.getfont(item_base.font).size or 655360
7353     d.width = crep.norule[1] * quad
7354     d.height = crep.norule[2] * quad
7355     d.depth = crep.norule[3] * quad
7356     head, new = node.insert_before(head, item, d)
7357
7358 elseif crep and crep.spacefactor then
7359     d = node.new(12, 13) -- (glue, spaceskip)
7360     local base_font = font.getfont(item_base.font)
7361     node.setglue(d,
7362                 crep.spacefactor[1] * base_font.parameters['space'],
7363                 crep.spacefactor[2] * base_font.parameters['space_stretch'],
7364                 crep.spacefactor[3] * base_font.parameters['space_shrink'])
7365     if mode == 0 then
7366         placeholder = ' '

```

```

7367         end
7368         head, new = node.insert_before(head, item, d)
7369
7370     elseif mode == 0 and crep and crep.space then
7371         -- ERROR
7372
7373     elseif crep and crep.kern then
7374         d = node.new(13, 1)      -- (kern, user)
7375         local quad = font.getfont(item_base.font).size or 655360
7376         d.attr = item_base.attr
7377         d.kern = crep.kern * quad
7378         head, new = node.insert_before(head, item, d)
7379
7380     elseif crep and crep.node then
7381         d = node.new(crep.node[1], crep.node[2])
7382         d.attr = item_base.attr
7383         head, new = node.insert_before(head, item, d)
7384
7385     end -- ie replacement cases
7386
7387     -- Shared by disc, space(factor), kern, node and penalty.
7388     if sc == 1 then
7389         word_head = head
7390     end
7391     if crep.insert then
7392         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7393         table.insert(w_nodes, sc, new)
7394         last = last + 1
7395     else
7396         w_nodes[sc] = d
7397         node.remove(head, item)
7398         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7399     end
7400
7401     last_match = utf8.offset(w, sc+1+step)
7402
7403     ::next::
7404
7405     end -- for each replacement
7406
7407     if Babel.debug then
7408         print('.....', '/')
7409         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7410     end
7411
7412     if dummy_node then
7413         node.remove(head, dummy_node)
7414         dummy_node = nil
7415     end
7416
7417     end -- for match
7418
7419     end -- for patterns
7420
7421     ::next::
7422     word_head = nw
7423 end -- for substring
7424 return head
7425 end
7426
7427 -- This table stores capture maps, numbered consecutively
7428 Babel.capture_maps = {}
7429

```

```

7430 -- The following functions belong to the next macro
7431 function Babel.capture_func(key, cap)
7432   local ret = "[" .. cap:gsub('{{[0-9]}}', "]]..m[%1]..[" .. "]"
7433   local cnt
7434   local u = unicode.utf8
7435   ret, cnt = ret:gsub('{{[0-9]}|([^\]]+)|([.-])}', Babel.capture_func_map)
7436   if cnt == 0 then
7437     ret = u.gsub(ret, '{{(%x%x%x%x+)}}',
7438       function (n)
7439         return u.char(tonumber(n, 16))
7440       end)
7441   end
7442   ret = ret:gsub("%[%[%]]%.%", '')
7443   ret = ret:gsub("%.%.%[%[%]]%", '')
7444   return key .. [[=function(m) return ]] .. ret .. [[ end]]
7445 end
7446
7447 function Babel.capt_map(from, mapno)
7448   return Babel.capture_maps[mapno][from] or from
7449 end
7450
7451 -- Handle the {n|abc|ABC} syntax in captures
7452 function Babel.capture_func_map(capno, from, to)
7453   local u = unicode.utf8
7454   from = u.gsub(from, '{{(%x%x%x%x+)}}',
7455     function (n)
7456       return u.char(tonumber(n, 16))
7457     end)
7458   to = u.gsub(to, '{{(%x%x%x%x+)}}',
7459     function (n)
7460       return u.char(tonumber(n, 16))
7461     end)
7462   local froms = {}
7463   for s in string.utfcharacters(from) do
7464     table.insert(froms, s)
7465   end
7466   local cnt = 1
7467   table.insert(Babel.capture_maps, {})
7468   local mlen = table.getn(Babel.capture_maps)
7469   for s in string.utfcharacters(to) do
7470     Babel.capture_maps[mlen][froms[cnt]] = s
7471     cnt = cnt + 1
7472   end
7473   return "]]..Babel.capt_map(m[" .. capno .. "], " ..
7474     (mlen) .. ").." .. "[["
7475 end
7476
7477 -- Create/Extend reversed sorted list of kashida weights:
7478 function Babel.capture_kashida(key, wt)
7479   wt = tonumber(wt)
7480   if Babel.kashida_wts then
7481     for p, q in ipairs(Babel.kashida_wts) do
7482       if wt == q then
7483         break
7484       elseif wt > q then
7485         table.insert(Babel.kashida_wts, p, wt)
7486         break
7487       elseif table.getn(Babel.kashida_wts) == p then
7488         table.insert(Babel.kashida_wts, wt)
7489       end
7490     end
7491   else
7492     Babel.kashida_wts = { wt }
7493   end

```

```

7493 end
7494 return 'kashida = ' .. wt
7495 end
7496
7497 function Babel.capture_node(id, subtype)
7498   local sbt = 0
7499   for k, v in pairs(node.subtypes(id)) do
7500     if v == subtype then sbt = k end
7501   end
7502   return 'node = {' .. node.id(id) .. ', ' .. sbt .. '}'
7503 end
7504
7505 -- Experimental: applies prehyphenation transforms to a string (letters
7506 -- and spaces).
7507 function Babel.string_prehyphenation(str, locale)
7508   local n, head, last, res
7509   head = node.new(8, 0) -- dummy (hack just to start)
7510   last = head
7511   for s in string.utfvalues(str) do
7512     if s == 20 then
7513       n = node.new(12, 0)
7514     else
7515       n = node.new(29, 0)
7516       n.char = s
7517     end
7518     node.set_attribute(n, Babel.attr_locale, locale)
7519     last.next = n
7520     last = n
7521   end
7522   head = Babel.hyphenate_replace(head, 0)
7523   res = ''
7524   for n in node.traverse(head) do
7525     if n.id == 12 then
7526       res = res .. ' '
7527     elseif n.id == 29 then
7528       res = res .. unicode.utf8.char(n.char)
7529     end
7530   end
7531   tex.print(res)
7532 end
7533 </transforms>

```

## 11.12 Lua: Auto bidi with basic and basic-r

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

```

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

```

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

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



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

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

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

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

```

7534 (*basic-r)
7535 Babel = Babel or {}
7536
7537 Babel.bidi_enabled = true
7538
7539 require('babel-data-bidi.lua')
7540
7541 local characters = Babel.characters
7542 local ranges = Babel.ranges
7543
7544 local DIR = node.id("dir")
7545
7546 local function dir_mark(head, from, to, outer)
7547   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
7548   local d = node.new(DIR)
7549   d.dir = '+' .. dir
7550   node.insert_before(head, from, d)
7551   d = node.new(DIR)
7552   d.dir = '-' .. dir
7553   node.insert_after(head, to, d)
7554 end
7555
7556 function Babel.bidi(head, ispar)
7557   local first_n, last_n          -- first and last char with nums
7558   local last_es                  -- an auxiliary 'last' used with nums
7559   local first_d, last_d          -- first and last char in L/R block
7560   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):

```

7561   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7562   local strong_lr = (strong == 'l') and 'l' or 'r'
7563   local outer = strong
7564
7565   local new_dir = false
7566   local first_dir = false
7567   local inmath = false
7568
7569   local last_lr
7570
7571   local type_n = ''
7572
7573   for item in node.traverse(head) do
7574

```

```

7575 -- three cases: glyph, dir, otherwise
7576 if item.id == node.id'glyph'
7577   or (item.id == 7 and item.subtype == 2) then
7578
7579   local itemchar
7580   if item.id == 7 and item.subtype == 2 then
7581     itemchar = item.replace.char
7582   else
7583     itemchar = item.char
7584   end
7585   local chardata = characters[itemchar]
7586   dir = chardata and chardata.d or nil
7587   if not dir then
7588     for nn, et in ipairs(ranges) do
7589       if itemchar < et[1] then
7590         break
7591       elseif itemchar <= et[2] then
7592         dir = et[3]
7593         break
7594       end
7595     end
7596   end
7597   dir = dir or 'l'
7598   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).

```

7599   if new_dir then
7600     attr_dir = 0
7601     for at in node.traverse(item.attr) do
7602       if at.number == Babel.attr_dir then
7603         attr_dir = at.value & 0x3
7604       end
7605     end
7606     if attr_dir == 1 then
7607       strong = 'r'
7608     elseif attr_dir == 2 then
7609       strong = 'al'
7610     else
7611       strong = 'l'
7612     end
7613     strong_lr = (strong == 'l') and 'l' or 'r'
7614     outer = strong_lr
7615     new_dir = false
7616   end
7617
7618   if dir == 'nsm' then dir = strong end -- W1

```

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

```

7619   dir_real = dir -- We need dir_real to set strong below
7620   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:

```

7621   if strong == 'al' then
7622     if dir == 'en' then dir = 'an' end -- W2
7623     if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7624     strong_lr = 'r' -- W3
7625   end

```

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

```

7626 elseif item.id == node.id'dir' and not inmath then
7627   new_dir = true
7628   dir = nil
7629 elseif item.id == node.id'math' then
7630   inmath = (item.subtype == 0)
7631 else
7632   dir = nil          -- Not a char
7633 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>.

```

7634 if dir == 'en' or dir == 'an' or dir == 'et' then
7635   if dir ~= 'et' then
7636     type_n = dir
7637   end
7638   first_n = first_n or item
7639   last_n = last_es or item
7640   last_es = nil
7641 elseif dir == 'es' and last_n then -- W3+W6
7642   last_es = item
7643 elseif dir == 'cs' then          -- it's right - do nothing
7644 elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7645   if strong_lr == 'r' and type_n ~= '' then
7646     dir_mark(head, first_n, last_n, 'r')
7647   elseif strong_lr == 'l' and first_d and type_n == 'an' then
7648     dir_mark(head, first_n, last_n, 'r')
7649     dir_mark(head, first_d, last_d, outer)
7650     first_d, last_d = nil, nil
7651   elseif strong_lr == 'l' and type_n ~= '' then
7652     last_d = last_n
7653   end
7654   type_n = ''
7655   first_n, last_n = nil, nil
7656 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:

```

7657 if dir == 'l' or dir == 'r' then
7658   if dir ~= outer then
7659     first_d = first_d or item
7660     last_d = item
7661   elseif first_d and dir ~= strong_lr then
7662     dir_mark(head, first_d, last_d, outer)
7663     first_d, last_d = nil, nil
7664   end
7665 end

```

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

```

7666 if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7667   item.char = characters[item.char] and
7668     characters[item.char].m or item.char
7669 elseif (dir or new_dir) and last_lr ~= item then
7670   local mir = outer .. strong_lr .. (dir or outer)
7671   if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7672     for ch in node.traverse(node.next(last_lr)) do
7673       if ch == item then break end

```

```

7674         if ch.id == node.id'glyph' and characters[ch.char] then
7675             ch.char = characters[ch.char].m or ch.char
7676         end
7677     end
7678 end
7679 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).

```

7680     if dir == 'l' or dir == 'r' then
7681         last_lr = item
7682         strong = dir_real          -- Don't search back - best save now
7683         strong_lr = (strong == 'l') and 'l' or 'r'
7684     elseif new_dir then
7685         last_lr = nil
7686     end
7687 end

```

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

```

7688     if last_lr and outer == 'r' then
7689         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7690             if characters[ch.char] then
7691                 ch.char = characters[ch.char].m or ch.char
7692             end
7693         end
7694     end
7695     if first_n then
7696         dir_mark(head, first_n, last_n, outer)
7697     end
7698     if first_d then
7699         dir_mark(head, first_d, last_d, outer)
7700     end

```

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

```

7701     return node.prev(head) or head
7702 end
7703 </basic-r>

```

And here the Lua code for bidi=basic:

```

7704 <(*basic)
7705 Babel = Babel or {}
7706
7707 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
7708
7709 Babel.fontmap = Babel.fontmap or {}
7710 Babel.fontmap[0] = {}          -- l
7711 Babel.fontmap[1] = {}          -- r
7712 Babel.fontmap[2] = {}          -- al/an
7713
7714 -- To cancel mirroring. Also OML, OMS, U?
7715 Babel.symbol_fonts = Babel.symbol_fonts or {}
7716 Babel.symbol_fonts[font.id('tenln')] = true
7717 Babel.symbol_fonts[font.id('tenlnw')] = true
7718 Babel.symbol_fonts[font.id('tencirc')] = true
7719 Babel.symbol_fonts[font.id('tencircw')] = true
7720
7721 Babel.bidi_enabled = true
7722 Babel.mirroring_enabled = true
7723
7724 require('babel-data-bidi.lua')
7725
7726 local characters = Babel.characters
7727 local ranges = Babel.ranges

```

```

7728
7729 local DIR = node.id('dir')
7730 local GLYPH = node.id('glyph')
7731
7732 local function insert_implicit(head, state, outer)
7733   local new_state = state
7734   if state.sim and state.eim and state.sim ~= state.eim then
7735     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
7736     local d = node.new(DIR)
7737     d.dir = '+' .. dir
7738     node.insert_before(head, state.sim, d)
7739     local d = node.new(DIR)
7740     d.dir = '-' .. dir
7741     node.insert_after(head, state.eim, d)
7742   end
7743   new_state.sim, new_state.eim = nil, nil
7744   return head, new_state
7745 end
7746
7747 local function insert_numeric(head, state)
7748   local new
7749   local new_state = state
7750   if state.san and state.ean and state.san ~= state.ean then
7751     local d = node.new(DIR)
7752     d.dir = '+TLT'
7753     _, new = node.insert_before(head, state.san, d)
7754     if state.san == state.sim then state.sim = new end
7755     local d = node.new(DIR)
7756     d.dir = '-TLT'
7757     _, new = node.insert_after(head, state.ean, d)
7758     if state.ean == state.eim then state.eim = new end
7759   end
7760   new_state.san, new_state.ean = nil, nil
7761   return head, new_state
7762 end
7763
7764 local function glyph_not_symbol_font(node)
7765   if node.id == GLYPH then
7766     return not Babel.symbol_fonts[node.font]
7767   else
7768     return false
7769   end
7770 end
7771
7772 -- TODO - \hbox with an explicit dir can lead to wrong results
7773 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7774 -- was made to improve the situation, but the problem is the 3-dir
7775 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7776 -- well.
7777
7778 function Babel.bidi(head, ispar, hdir)
7779   local d -- d is used mainly for computations in a loop
7780   local prev_d = ''
7781   local new_d = false
7782
7783   local nodes = {}
7784   local outer_first = nil
7785   local inmath = false
7786
7787   local glue_d = nil
7788   local glue_i = nil
7789
7790   local has_en = false

```

```

7791 local first_et = nil
7792
7793 local has_hyperlink = false
7794
7795 local ATDIR = Babel.attr_dir
7796 local attr_d
7797
7798 local save_outer
7799 local temp = node.get_attribute(head, ATDIR)
7800 if temp then
7801     temp = temp & 0x3
7802     save_outer = (temp == 0 and 'l') or
7803                 (temp == 1 and 'r') or
7804                 (temp == 2 and 'al')
7805 elseif ispar then -- Or error? Shouldn't happen
7806     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7807 else -- Or error? Shouldn't happen
7808     save_outer = ('TRT' == hdir) and 'r' or 'l'
7809 end
7810 -- when the callback is called, we are just _after_ the box,
7811 -- and the textdir is that of the surrounding text
7812 -- if not ispar and hdir ~= tex.textdir then
7813 --     save_outer = ('TRT' == hdir) and 'r' or 'l'
7814 -- end
7815 local outer = save_outer
7816 local last = outer
7817 -- 'al' is only taken into account in the first, current loop
7818 if save_outer == 'al' then save_outer = 'r' end
7819
7820 local fontmap = Babel.fontmap
7821
7822 for item in node.traverse(head) do
7823
7824     -- In what follows, #node is the last (previous) node, because the
7825     -- current one is not added until we start processing the neutrals.
7826
7827     -- three cases: glyph, dir, otherwise
7828     if glyph_not_symbol_font(item)
7829         or (item.id == 7 and item.subtype == 2) then
7830
7831         if node.get_attribute(item, ATDIR) == 128 then goto nextnode end
7832
7833         local d_font = nil
7834         local item_r
7835         if item.id == 7 and item.subtype == 2 then
7836             item_r = item.replace -- automatic discs have just 1 glyph
7837         else
7838             item_r = item
7839         end
7840
7841         local chardata = characters[item_r.char]
7842         d = chardata and chardata.d or nil
7843         if not d or d == 'nsm' then
7844             for nn, et in ipairs(ranges) do
7845                 if item_r.char < et[1] then
7846                     break
7847                 elseif item_r.char <= et[2] then
7848                     if not d then d = et[3]
7849                     elseif d == 'nsm' then d_font = et[3]
7850                     end
7851                     break
7852                 end
7853             end
7854         end
7855     end

```

```

7854     end
7855     d = d or 'l'
7856
7857     -- A short 'pause' in bidi for mapfont
7858     d_font = d_font or d
7859     d_font = (d_font == 'l' and 0) or
7860             (d_font == 'nsm' and 0) or
7861             (d_font == 'r' and 1) or
7862             (d_font == 'al' and 2) or
7863             (d_font == 'an' and 2) or nil
7864     if d_font and fontmap and fontmap[d_font][item_r.font] then
7865         item_r.font = fontmap[d_font][item_r.font]
7866     end
7867
7868     if new_d then
7869         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7870         if inmath then
7871             attr_d = 0
7872         else
7873             attr_d = node.get_attribute(item, ATDIR)
7874             attr_d = attr_d & 0x3
7875         end
7876         if attr_d == 1 then
7877             outer_first = 'r'
7878             last = 'r'
7879         elseif attr_d == 2 then
7880             outer_first = 'r'
7881             last = 'al'
7882         else
7883             outer_first = 'l'
7884             last = 'l'
7885         end
7886         outer = last
7887         has_en = false
7888         first_et = nil
7889         new_d = false
7890     end
7891
7892     if glue_d then
7893         if (d == 'l' and 'l' or 'r') ~= glue_d then
7894             table.insert(nodes, {glue_i, 'on', nil})
7895         end
7896         glue_d = nil
7897         glue_i = nil
7898     end
7899
7900     elseif item.id == DIR then
7901         d = nil
7902
7903         if head ~= item then new_d = true end
7904
7905     elseif item.id == node.id'glue' and item.subtype == 13 then
7906         glue_d = d
7907         glue_i = item
7908         d = nil
7909
7910     elseif item.id == node.id'math' then
7911         inmath = (item.subtype == 0)
7912
7913     elseif item.id == 8 and item.subtype == 19 then
7914         has_hyperlink = true
7915
7916     else

```

```

7917     d = nil
7918 end
7919
7920 -- AL <= EN/ET/ES      -- W2 + W3 + W6
7921 if last == 'al' and d == 'en' then
7922     d = 'an'           -- W3
7923 elseif last == 'al' and (d == 'et' or d == 'es') then
7924     d = 'on'           -- W6
7925 end
7926
7927 -- EN + CS/ES + EN      -- W4
7928 if d == 'en' and #nodes >= 2 then
7929     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7930         and nodes[#nodes-1][2] == 'en' then
7931         nodes[#nodes][2] = 'en'
7932     end
7933 end
7934
7935 -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
7936 if d == 'an' and #nodes >= 2 then
7937     if (nodes[#nodes][2] == 'cs')
7938         and nodes[#nodes-1][2] == 'an' then
7939         nodes[#nodes][2] = 'an'
7940     end
7941 end
7942
7943 -- ET/EN                  -- W5 + W7->l / W6->on
7944 if d == 'et' then
7945     first_et = first_et or (#nodes + 1)
7946 elseif d == 'en' then
7947     has_en = true
7948     first_et = first_et or (#nodes + 1)
7949 elseif first_et then      -- d may be nil here !
7950     if has_en then
7951         if last == 'l' then
7952             temp = 'l'    -- W7
7953         else
7954             temp = 'en'   -- W5
7955         end
7956     else
7957         temp = 'on'      -- W6
7958     end
7959     for e = first_et, #nodes do
7960         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
7961     end
7962     first_et = nil
7963     has_en = false
7964 end
7965
7966 -- Force mathdir in math if ON (currently works as expected only
7967 -- with 'l')
7968
7969 if inmath and d == 'on' then
7970     d = ('TRT' == tex.mathdir) and 'r' or 'l'
7971 end
7972
7973 if d then
7974     if d == 'al' then
7975         d = 'r'
7976         last = 'al'
7977     elseif d == 'l' or d == 'r' then
7978         last = d
7979     end

```



```

7980     prev_d = d
7981     table.insert(nodes, {item, d, outer_first})
7982 end
7983
7984 node.set_attribute(item, ATDIR, 128)
7985 outer_first = nil
7986
7987 ::nextnode::
7988
7989 end -- for each node
7990
7991 -- TODO -- repeated here in case EN/ET is the last node. Find a
7992 -- better way of doing things:
7993 if first_et then      -- dir may be nil here !
7994     if has_en then
7995         if last == 'l' then
7996             temp = 'l'    -- W7
7997         else
7998             temp = 'en'   -- W5
7999         end
8000     else
8001         temp = 'on'      -- W6
8002     end
8003     for e = first_et, #nodes do
8004         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8005     end
8006 end
8007
8008 -- dummy node, to close things
8009 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8010
8011 ----- NEUTRAL -----
8012
8013 outer = save_outer
8014 last = outer
8015
8016 local first_on = nil
8017
8018 for q = 1, #nodes do
8019     local item
8020
8021     local outer_first = nodes[q][3]
8022     outer = outer_first or outer
8023     last = outer_first or last
8024
8025     local d = nodes[q][2]
8026     if d == 'an' or d == 'en' then d = 'r' end
8027     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
8028
8029     if d == 'on' then
8030         first_on = first_on or q
8031     elseif first_on then
8032         if last == d then
8033             temp = d
8034         else
8035             temp = outer
8036         end
8037         for r = first_on, q - 1 do
8038             nodes[r][2] = temp
8039             item = nodes[r][1]    -- MIRRORING
8040             if Babel.mirroring_enabled and glyph_not_symbol_font(item)
8041                 and temp == 'r' and characters[item.char] then
8042                 local font_mode = ''

```

```

8043         if item.font > 0 and font.fonts[item.font].properties then
8044             font_mode = font.fonts[item.font].properties.mode
8045         end
8046         if font_mode ~= 'harf' and font_mode ~= 'plug' then
8047             item.char = characters[item.char].m or item.char
8048         end
8049     end
8050 end
8051 first_on = nil
8052 end
8053
8054 if d == 'r' or d == 'l' then last = d end
8055 end
8056
8057 ----- IMPLICIT, REORDER -----
8058
8059 outer = save_outer
8060 last = outer
8061
8062 local state = {}
8063 state.has_r = false
8064
8065 for q = 1, #nodes do
8066
8067     local item = nodes[q][1]
8068
8069     outer = nodes[q][3] or outer
8070
8071     local d = nodes[q][2]
8072
8073     if d == 'nsm' then d = last end          -- W1
8074     if d == 'en' then d = 'an' end
8075     local isdir = (d == 'r' or d == 'l')
8076
8077     if outer == 'l' and d == 'an' then
8078         state.san = state.san or item
8079         state.ean = item
8080     elseif state.san then
8081         head, state = insert_numeric(head, state)
8082     end
8083
8084     if outer == 'l' then
8085         if d == 'an' or d == 'r' then      -- im -> implicit
8086             if d == 'r' then state.has_r = true end
8087             state.sim = state.sim or item
8088             state.eim = item
8089         elseif d == 'l' and state.sim and state.has_r then
8090             head, state = insert_implicit(head, state, outer)
8091         elseif d == 'l' then
8092             state.sim, state.eim, state.has_r = nil, nil, false
8093         end
8094     else
8095         if d == 'an' or d == 'l' then
8096             if nodes[q][3] then -- nil except after an explicit dir
8097                 state.sim = item -- so we move sim 'inside' the group
8098             else
8099                 state.sim = state.sim or item
8100             end
8101             state.eim = item
8102         elseif d == 'r' and state.sim then
8103             head, state = insert_implicit(head, state, outer)
8104         elseif d == 'r' then
8105             state.sim, state.eim = nil, nil

```

```

8106     end
8107 end
8108
8109 if isdir then
8110     last = d          -- Don't search back - best save now
8111 elseif d == 'on' and state.san then
8112     state.san = state.san or item
8113     state.ean = item
8114 end
8115
8116 end
8117
8118 head = node.prev(head) or head
8119
8120 ----- FIX HYPERLINKS -----
8121
8122 if has_hyperlink then
8123     local flag, linking = 0, 0
8124     for item in node.traverse(head) do
8125         if item.id == DIR then
8126             if item.dir == '+TRT' or item.dir == '+TLT' then
8127                 flag = flag + 1
8128             elseif item.dir == '-TRT' or item.dir == '-TLT' then
8129                 flag = flag - 1
8130             end
8131             elseif item.id == 8 and item.subtype == 19 then
8132                 linking = flag
8133             elseif item.id == 8 and item.subtype == 20 then
8134                 if linking > 0 then
8135                     if item.prev.id == DIR and
8136                        (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
8137                         d = node.new(DIR)
8138                         d.dir = item.prev.dir
8139                         node.remove(head, item.prev)
8140                         node.insert_after(head, item, d)
8141                     end
8142                 end
8143                 linking = 0
8144             end
8145         end
8146     end
8147
8148     return head
8149 end
8150 -- Make sure anything is marked as 'bidi done' (including nodes inserted
8151 -- after the babel algorithm).
8152 function Babel.unset_atdir(head)
8153     local ATDIR = Babel.attr_dir
8154     for item in node.traverse(head) do
8155         node.set_attribute(item, ATDIR, 128)
8156     end
8157     return head
8158 end
8159 /basic

```

## 12 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.

## 13 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.

```
8160 <*nil>
8161 \ProvidesLanguage{nil}[<@date@> v<@version@> Nil language]
8162 \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.

```
8163 \ifx\l@nil\undefined
8164 \newlanguage\l@nil
8165 \@namedef{bbl@hyphendata@the\l@nil}{}}}% Remove warning
8166 \let\bbl@elt\relax
8167 \edef\bbl@languages{% Add it to the list of languages
8168 \bbl@languages\bbl@elt{nil}{the\l@nil}{}}
8169 \fi
```

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

```
8170 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

### **\captionnil**

#### **\datenil**

```
8171 \let\captionnil\@empty
8172 \let\datenil\@empty
```

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

```
8173 \def\bbl@inidata@nil{%
8174 \bbl@elt{identification}{tag.ini}{und}%
8175 \bbl@elt{identification}{load.level}{0}%
8176 \bbl@elt{identification}{charset}{utf8}%
8177 \bbl@elt{identification}{version}{1.0}%
8178 \bbl@elt{identification}{date}{2022-05-16}%
8179 \bbl@elt{identification}{name.local}{nil}%
8180 \bbl@elt{identification}{name.english}{nil}%
8181 \bbl@elt{identification}{name.babel}{nil}%
8182 \bbl@elt{identification}{tag.bcp47}{und}%
8183 \bbl@elt{identification}{language.tag.bcp47}{und}%
8184 \bbl@elt{identification}{tag.opentype}{dflt}%
8185 \bbl@elt{identification}{script.name}{Latin}%
8186 \bbl@elt{identification}{script.tag.bcp47}{Latn}%
8187 \bbl@elt{identification}{script.tag.opentype}{DFLT}%
8188 \bbl@elt{identification}{level}{1}%
8189 \bbl@elt{identification}{encodings}{}%
8190 \bbl@elt{identification}{derivate}{no}}
8191 \@namedef{bbl@tbc@nil}{und}
8192 \@namedef{bbl@lbc@nil}{und}
```

```

8193 \@namedef{bbl@casing@nil}{und} % TODO
8194 \@namedef{bbl@lotf@nil}{dflt}
8195 \@namedef{bbl@elname@nil}{nil}
8196 \@namedef{bbl@lname@nil}{nil}
8197 \@namedef{bbl@esname@nil}{Latin}
8198 \@namedef{bbl@sname@nil}{Latin}
8199 \@namedef{bbl@sbcpr@nil}{Latn}
8200 \@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.

```

8201 \ldf@finish{nil}
8202 </nil>

```

## 14 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.

```

8203 <<*Compute Julian day>> ≡
8204 \def\bbl@fpmmod#1#2{(#1-#2*floor(#1/#2))}
8205 \def\bbl@cs@gregleap#1{%
8206   (\bbl@fpmmod{#1}{4} == 0) &&
8207   (!((\bbl@fpmmod{#1}{100} == 0) && (\bbl@fpmmod{#1}{400} != 0)))}
8208 \def\bbl@cs@jd#1#2#3{% year, month, day
8209   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
8210     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
8211     floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) +
8212     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3 } }
8213 <</Compute Julian day>>

```

### 14.1 Islamic

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

```

8214 <*ca-islamic>
8215 \ExplSyntaxOn
8216 <@Compute Julian day@>
8217 % == islamic (default)
8218 % Not yet implemented
8219 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{}

```

The Civil calendar.

```

8220 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
8221   ((#3 + ceil(29.5 * (#2 - 1)) +
8222     (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
8223     1948439.5) - 1) }
8224 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
8225 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
8226 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
8227 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
8228 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
8229 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
8230   \edef\bbl@tempa{%
8231     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
8232   \edef#5{%
8233     \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
8234   \edef#6{\fp_eval:n{
8235     min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
8236   \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).

```

8237 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
8238 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
8239 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
8240 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
8241 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
8242 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
8243 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
8244 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
8245 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
8246 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
8247 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
8248 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
8249 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
8250 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8251 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8252 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8253 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
8254 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
8255 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8256 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8257 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8258 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8259 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8260 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
8261 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8262 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8263 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
8264 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8265 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8266 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8267 65401,65431,65460,65490,65520}
8268 \namedef\bbl@ca@islamic-umalqura+{\bbl@ca@islamcuqr@x{+1}}
8269 \namedef\bbl@ca@islamic-umalqura-\bbl@ca@islamcuqr@x{-1}}
8270 \namedef\bbl@ca@islamic-umalqura-{\bbl@ca@islamcuqr@x{-1}}
8271 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
8272 \ifnum#2>2014 \ifnum#2<2038
8273 \bbl@afterfi\expandafter\@gobble
8274 \fi\fi
8275 {\bbl@error{year-out-range}{2014-2038}}{}}%
8276 \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
8277 \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8278 \count@\@ne
8279 \bbl@foreach\bbl@cs@umalqura@data{%
8280 \advance\count@\@ne
8281 \ifnum##1>\bbl@tempd\else
8282 \edef\bbl@tempe{\the\count@}%
8283 \edef\bbl@tempb{##1}%
8284 \fi}%
8285 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
8286 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1 ) / 12) }}% annus
8287 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
8288 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
8289 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
8290 \ExplSyntaxOff
8291 \bbl@add\bbl@precalendar{%
8292 \bbl@replace\bbl@ld@calendar{-civil}}}%
8293 \bbl@replace\bbl@ld@calendar{-umalqura}}}%
8294 \bbl@replace\bbl@ld@calendar{+}}}%
8295 \bbl@replace\bbl@ld@calendar{-}}}%

```

8296 ⟨/ca-islamic⟩

## 14.2 Hebrew

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

```
8297 ⟨*ca-hebrew⟩
8298 \newcount\bbl@cntcommon
8299 \def\bbl@remainder#1#2#3{%
8300   #3=#1\relax
8301   \divide #3 by #2\relax
8302   \multiply #3 by -#2\relax
8303   \advance #3 by #1\relax}%
8304 \newif\ifbbl@divisible
8305 \def\bbl@checkifdivisible#1#2{%
8306   {\countdef\tmp=0
8307    \bbl@remainder{#1}{#2}{\tmp}%
8308    \ifnum \tmp=0
8309      \global\bbl@divisibletrue
8310    \else
8311      \global\bbl@divisiblefalse
8312    \fi}}
8313 \newif\ifbbl@gregleap
8314 \def\bbl@ifgregleap#1{%
8315   \bbl@checkifdivisible{#1}{4}%
8316   \ifbbl@divisible
8317     \bbl@checkifdivisible{#1}{100}%
8318     \ifbbl@divisible
8319       \bbl@checkifdivisible{#1}{400}%
8320       \ifbbl@divisible
8321         \bbl@gregleaptrue
8322       \else
8323         \bbl@gregleapfalse
8324       \fi
8325     \else
8326       \bbl@gregleaptrue
8327     \fi
8328   \else
8329     \bbl@gregleapfalse
8330   \fi
8331   \ifbbl@gregleap}
8332 \def\bbl@gregdayspriormonths#1#2#3{%
8333   {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
8334     181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
8335   \bbl@ifgregleap{#2}%
8336   \ifnum #1 > 2
8337     \advance #3 by 1
8338   \fi
8339   \fi
8340   \global\bbl@cntcommon=#3}%
8341   #3=\bbl@cntcommon}
8342 \def\bbl@gregdaysprioryears#1#2{%
8343   {\countdef\tmpc=4
8344    \countdef\tmpb=2
8345    \tmpb=#1\relax
8346    \advance \tmpb by -1
8347    \tmpc=\tmpb
8348    \multiply \tmpc by 365
8349    #2=\tmpc
8350    \tmpc=\tmpb
8351    \divide \tmpc by 4
8352    \advance #2 by \tmpc
```

```

8353 \tmpc=\tmpb
8354 \divide \tmpc by 100
8355 \advance #2 by -\tmpc
8356 \tmpc=\tmpb
8357 \divide \tmpc by 400
8358 \advance #2 by \tmpc
8359 \global\bbl@cntcommon=#2\relax}%
8360 #2=\bbl@cntcommon}
8361 \def\bbl@absfromgreg#1#2#3#4{%
8362 {\countdef\tmpd=0
8363 #4=#1\relax
8364 \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
8365 \advance #4 by \tmpd
8366 \bbl@gregdaysprioryears{#3}{\tmpd}%
8367 \advance #4 by \tmpd
8368 \global\bbl@cntcommon=#4\relax}%
8369 #4=\bbl@cntcommon}
8370 \newif\ifbbl@hebrleap
8371 \def\bbl@checkleaphebryear#1{%
8372 {\countdef\tmpa=0
8373 \countdef\tmpb=1
8374 \tmpa=#1\relax
8375 \multiply \tmpa by 7
8376 \advance \tmpa by 1
8377 \bbl@remainder{\tmpa}{19}{\tmpb}%
8378 \ifnum \tmpb < 7
8379 \global\bbl@hebrleaptrue
8380 \else
8381 \global\bbl@hebrleapfalse
8382 \fi}}
8383 \def\bbl@hebrlapsedmonths#1#2{%
8384 {\countdef\tmpa=0
8385 \countdef\tmpb=1
8386 \countdef\tmpc=2
8387 \tmpa=#1\relax
8388 \advance \tmpa by -1
8389 #2=\tmpa
8390 \divide #2 by 19
8391 \multiply #2 by 235
8392 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8393 \tmpc=\tmpb
8394 \multiply \tmpb by 12
8395 \advance #2 by \tmpb
8396 \multiply \tmpc by 7
8397 \advance \tmpc by 1
8398 \divide \tmpc by 19
8399 \advance #2 by \tmpc
8400 \global\bbl@cntcommon=#2}%
8401 #2=\bbl@cntcommon}
8402 \def\bbl@hebrlapseddays#1#2{%
8403 {\countdef\tmpa=0
8404 \countdef\tmpb=1
8405 \countdef\tmpc=2
8406 \bbl@hebrlapsedmonths{#1}{#2}%
8407 \tmpa=#2\relax
8408 \multiply \tmpa by 13753
8409 \advance \tmpa by 5604
8410 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8411 \divide \tmpa by 25920
8412 \multiply #2 by 29
8413 \advance #2 by 1
8414 \advance #2 by \tmpa
8415 \bbl@remainder{#2}{7}{\tmpa}%

```



```

8416 \ifnum \tmpc < 19440
8417     \ifnum \tmpc < 9924
8418     \else
8419         \ifnum \tmpa=2
8420             \bbl@checkleaphebrewyear{#1}% of a common year
8421             \ifbbl@hebrleap
8422             \else
8423                 \advance #2 by 1
8424             \fi
8425         \fi
8426     \fi
8427     \ifnum \tmpc < 16789
8428     \else
8429         \ifnum \tmpa=1
8430             \advance #1 by -1
8431             \bbl@checkleaphebrewyear{#1}% at the end of leap year
8432             \ifbbl@hebrleap
8433                 \advance #2 by 1
8434             \fi
8435         \fi
8436     \fi
8437 \else
8438     \advance #2 by 1
8439 \fi
8440 \bbl@remainder{#2}{7}{\tmpa}%
8441 \ifnum \tmpa=0
8442     \advance #2 by 1
8443 \else
8444     \ifnum \tmpa=3
8445         \advance #2 by 1
8446     \else
8447         \ifnum \tmpa=5
8448             \advance #2 by 1
8449         \fi
8450     \fi
8451 \fi
8452 \global\bbl@cntcommon=#2\relax}%
8453 #2=\bbl@cntcommon}
8454 \def\bbl@daysinhebrewyear#1#2{%
8455     {\countdef\tmpe=12
8456     \bbl@hebreleapseddays{#1}{\tmpe}%
8457     \advance #1 by 1
8458     \bbl@hebreleapseddays{#1}{#2}%
8459     \advance #2 by -\tmpe
8460     \global\bbl@cntcommon=#2}%
8461 #2=\bbl@cntcommon}
8462 \def\bbl@hebrdayspriormonths#1#2#3{%
8463     {\countdef\tmpf= 14
8464     #3=\ifcase #1\relax
8465         0 \or
8466         0 \or
8467         30 \or
8468         59 \or
8469         89 \or
8470         118 \or
8471         148 \or
8472         148 \or
8473         177 \or
8474         207 \or
8475         236 \or
8476         266 \or
8477         295 \or
8478         325 \or

```

```

8479         400
8480     \fi
8481     \bbl@checkleaphebrewyear{#2}%
8482     \ifbbl@hebrleap
8483         \ifnum #1 > 6
8484             \advance #3 by 30
8485         \fi
8486     \fi
8487     \bbl@daysinhebrewyear{#2}{\tmpf}%
8488     \ifnum #1 > 3
8489         \ifnum \tmpf=353
8490             \advance #3 by -1
8491         \fi
8492         \ifnum \tmpf=383
8493             \advance #3 by -1
8494         \fi
8495     \fi
8496     \ifnum #1 > 2
8497         \ifnum \tmpf=355
8498             \advance #3 by 1
8499         \fi
8500         \ifnum \tmpf=385
8501             \advance #3 by 1
8502         \fi
8503     \fi
8504     \global\bbl@cntcommon=#3\relax}%
8505     #3=\bbl@cntcommon}
8506 \def\bbl@absfromhebr#1#2#3#4{%
8507     {#4=#1\relax
8508     \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8509     \advance #4 by #1\relax
8510     \bbl@hebreleapseddays{#3}{#1}%
8511     \advance #4 by #1\relax
8512     \advance #4 by -1373429
8513     \global\bbl@cntcommon=#4\relax}%
8514     #4=\bbl@cntcommon}
8515 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8516     {\countdef\tmpx= 17
8517     \countdef\tmpy= 18
8518     \countdef\tmpz= 19
8519     #6=#3\relax
8520     \global\advance #6 by 3761
8521     \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8522     \tmpz=1 \tmpy=1
8523     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8524     \ifnum \tmpx > #4\relax
8525         \global\advance #6 by -1
8526         \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8527     \fi
8528     \advance #4 by -\tmpx
8529     \advance #4 by 1
8530     #5=#4\relax
8531     \divide #5 by 30
8532     \loop
8533         \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8534         \ifnum \tmpx < #4\relax
8535             \advance #5 by 1
8536             \tmpy=\tmpx
8537         \repeat
8538     \global\advance #5 by -1
8539     \global\advance #4 by -\tmpy}}
8540 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebrewyear
8541 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear

```

```

8542 \def\bbl@ca@hebrew#1-#2-#3\@@#4#5#6{%
8543 \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8544 \bbl@hebrfromgreg
8545 {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8546 {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
8547 \edef#4{\the\bbl@hebryear}%
8548 \edef#5{\the\bbl@hebrmonth}%
8549 \edef#6{\the\bbl@hebrday}}
8550 </ca-hebrew>

```

### 14.3 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).

```

8551 <ca-persian>
8552 \ExplSyntaxOn
8553 <@Compute Julian day@>
8554 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8555 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8556 \def\bbl@ca@persian#1-#2-#3\@@#4#5#6{%
8557 \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8558 \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8559 \bbl@afterfi\expandafter\@gobble
8560 \fi\fi
8561 {\bbl@error{year-out-range}{2013-2050}{}}}%
8562 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8563 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8564 \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8565 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8566 \ifnum\bbl@tempc<\bbl@tempb
8567 \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8568 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8569 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8570 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8571 \fi
8572 \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8573 \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8574 \edef#5{\fp_eval:n{% set Jalali month
8575 (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8576 \edef#6{\fp_eval:n{% set Jalali day
8577 (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6))}}}%
8578 \ExplSyntaxOff
8579 </ca-persian>

```

### 14.4 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.

```

8580 <ca-coptic>
8581 \ExplSyntaxOn
8582 <@Compute Julian day@>
8583 \def\bbl@ca@coptic#1-#2-#3\@@#4#5#6{%
8584 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8585 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8586 \edef#4{\fp_eval:n{%
8587 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8588 \edef\bbl@tempc{\fp_eval:n{%
8589 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8590 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%

```

```

8591 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}
8592 \ExplSyntaxOff
8593 </ca-coptic>
8594 <*ca-ethiopic>
8595 \ExplSyntaxOn
8596 <@Compute Julian day@>
8597 \def\bbl@ca@ethiopic#1-#2-#3\@@#4#5#6{%
8598 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8599 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8600 \edef#4{\fp_eval:n{%
8601 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8602 \edef\bbl@tempc{\fp_eval:n{%
8603 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8604 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8605 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}
8606 \ExplSyntaxOff
8607 </ca-ethiopic>

```

## 14.5 Buddhist

That's very simple.

```

8608 <*ca-buddhist>
8609 \def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
8610 \edef#4{\number\numexpr#1+543\relax}%
8611 \edef#5{#2}%
8612 \edef#6{#3}}
8613 </ca-buddhist>
8614 %
8615 % \subsection{Chinese}
8616 %
8617 % Brute force, with the Julian day of first day of each month. The
8618 % table has been computed with the help of \textsf{python-lunardate} by
8619 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
8620 % is 2015-2044.
8621 %
8622 % \begin{macrocode}
8623 <*ca-chinese>
8624 \ExplSyntaxOn
8625 <@Compute Julian day@>
8626 \def\bbl@ca@chinese#1-#2-#3\@@#4#5#6{%
8627 \edef\bbl@tempd{\fp_eval:n{%
8628 \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
8629 \count@ \z@
8630 \@tempcnta=2015
8631 \bbl@foreach\bbl@cs@chinese@data{%
8632 \ifnum##1>\bbl@tempd\else
8633 \advance\count@\@ne
8634 \ifnum\count@>12
8635 \count@\@ne
8636 \advance\@tempcnta\@ne\fi
8637 \bbl@xin@{,##1,}{,\bbl@cs@chinese@leap,}%
8638 \ifin@
8639 \advance\count@\m@ne
8640 \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
8641 \else
8642 \edef\bbl@tempe{\the\count@}%
8643 \fi
8644 \edef\bbl@tempb{##1}%
8645 \fi}%
8646 \edef#4{\the\@tempcnta}%
8647 \edef#5{\bbl@tempe}%
8648 \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
8649 \def\bbl@cs@chinese@leap{%

```

```

8650 885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
8651 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
8652 354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
8653 768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
8654 1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
8655 1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
8656 1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
8657 2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
8658 2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
8659 2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
8660 3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
8661 3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
8662 3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
8663 4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
8664 4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
8665 5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
8666 5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
8667 5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
8668 6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
8669 6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
8670 6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
8671 7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
8672 7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
8673 7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
8674 8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8675 8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8676 8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
8677 9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
8678 9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
8679 10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
8680 10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
8681 10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
8682 10896,10926,10956,10986,11015,11045,11074,11103}
8683 \ExplSyntaxOff
8684 </ca-chinese>

```

## 15 Support for Plain T<sub>E</sub>X (plain.def)

### 15.1 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<sub>E</sub>X-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`.

```

8685 <*bplain | blplain>
8686 \catcode`\{=1 % left brace is begin-group character
8687 \catcode`\}=2 % right brace is end-group character
8688 \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).

```

8689 \openin 0 hyphen.cfg
8690 \ifeof0
8691 \else
8692   \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.

```

8693 \def\input #1 {%
8694   \let\input\input
8695   \a hyphen.cfg
8696   \let\input\input
8697 }
8698 \fi
8699 </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`.

```

8700 <bplain>\a plain.tex
8701 <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.

```

8702 <bplain>\def\fmtname{babel-plain}
8703 <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.

## 15.2 Emulating some $\text{\LaTeX}$ features

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

```

8704 <<*Emulate LaTeX>> ≡
8705 \def\@empty{}
8706 \def\loadlocalcfg#1{%
8707   \openin0#1.cfg
8708   \ifeof0
8709     \closein0
8710   \else
8711     \closein0
8712     {\immediate\writel6{*****}%
8713      \immediate\writel6{* Local config file #1.cfg used}%
8714      \immediate\writel6{*}%
8715     }
8716     \input #1.cfg\relax
8717   \fi
8718   \@endofldf}

```

## 15.3 General tools

A number of  $\text{\LaTeX}$  macro's that are needed later on.

```

8719 \long\def\@firstofone#1{#1}
8720 \long\def\@firstoftwo#1#2{#1}
8721 \long\def\@secondoftwo#1#2{#2}
8722 \def\@nnil{\@nil}
8723 \def\@gobbletwo#1#2{}
8724 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
8725 \def\@staror@long#1{%
8726   \@ifstar
8727   {\let\@ngrel@x\relax#1}%

```

```

8728 {\let\l@ngrel@x\long#1}}
8729 \let\l@ngrel@x\relax
8730 \def\@car#1#2\@nil{#1}
8731 \def\@cdr#1#2\@nil{#2}
8732 \let\@typeset@protect\relax
8733 \let\protected@edef\edef
8734 \long\def\@gobble#1{}
8735 \edef\@backslashchar{\expandafter\@gobble\string\}
8736 \def\strip@prefix#1>{}
8737 \def\g@addto@macro#1#2{%
8738     \toks@\expandafter{#1#2}%
8739     \xdef#1{\the\toks@}}
8740 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8741 \def\@nameuse#1{\csname #1\endcsname}
8742 \def\@ifundefined#1{%
8743     \expandafter\ifx\csname#1\endcsname\relax
8744     \expandafter\@firstoftwo
8745     \else
8746     \expandafter\@secondoftwo
8747     \fi}
8748 \def\@expandtwoargs#1#2#3{%
8749     \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8750 \def\zap@space#1 #2{%
8751     #1%
8752     \ifx#2\@empty\else\expandafter\zap@space\fi
8753     #2}
8754 \let\bbl@trace\@gobble
8755 \def\bbl@error#1{% Implicit #2#3#4
8756     \begingroup
8757     \catcode`\=0 \catcode`\==12 \catcode`\`=12
8758     \catcode`\^M=5 \catcode`\%=14
8759     \input errbabel.def
8760     \endgroup
8761     \bbl@error{#1}}
8762 \def\bbl@warning#1{%
8763     \begingroup
8764     \newlinechar=`^^J
8765     \def\{^^J(babel) }%
8766     \message{\#1}%
8767     \endgroup}
8768 \let\bbl@infowarn\bbl@warning
8769 \def\bbl@info#1{%
8770     \begingroup
8771     \newlinechar=`^^J
8772     \def\{^^J}%
8773     \wlog{#1}%
8774     \endgroup}

```

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

```

8775 \ifx\@preamblecmds\undefined
8776 \def\@preamblecmds{}
8777 \fi
8778 \def\@onlypreamble#1{%
8779     \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8780         \@preamblecmds\do#1}
8781 \onlypreamble\@onlypreamble

```

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

```

8782 \def\begindocument{%
8783     \@begindocumenthook
8784     \global\let\@begindocumenthook\undefined
8785     \def\do##1{\global\let##1\@undefined}%
8786     \@preamblecmds

```

```

8787 \global\let\do\noexpand}

8788 \ifx\@begindocumenthook\@undefined
8789 \def\@begindocumenthook{}
8790 \fi
8791 \@onlypreamble\@begindocumenthook
8792 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

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

```

8793 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
8794 \@onlypreamble\AtEndOfPackage
8795 \def\@endofldf{}
8796 \@onlypreamble\@endofldf
8797 \let\bbl@afterlang\@empty
8798 \chardef\bbl@opt@hyphenmap\z@

```

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

```

8799 \catcode`\&=\z@
8800 \ifx&\if@files\@undefined
8801 \expandafter\let\csname if@files\expandafter\endcsname
8802 \csname iffalse\endcsname
8803 \fi
8804 \catcode`\&=4

```

Mimic  $\LaTeX$ 's commands to define control sequences.

```

8805 \def\newcommand{\@star@or@long\new@command}
8806 \def\new@command#1{%
8807 \@testopt{\@newcommand#1}0}
8808 \def\@newcommand#1[#2]{%
8809 \@ifnextchar [{\@xargdef#1[#2]}%
8810 {\@argdef#1[#2]}}
8811 \long\def\@argdef#1[#2]#3{%
8812 \@yargdef#1\@ne{#2}{#3}}
8813 \long\def\@xargdef#1[#2][#3]#4{%
8814 \expandafter\def\expandafter#1\expandafter{%
8815 \expandafter\@protected@testopt\expandafter #1%
8816 \csname\string#1\expandafter\endcsname{#3}}}%
8817 \expandafter\@yargdef \csname\string#1\endcsname
8818 \tw@{#2}{#4}}
8819 \long\def\@yargdef#1#2#3{%
8820 \@tempcnta#3\relax
8821 \advance \@tempcnta \@ne
8822 \let\@hash@\relax
8823 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
8824 \@tempcntb #2%
8825 \@whilenum \@tempcntb < \@tempcnta
8826 \do{%
8827 \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
8828 \advance \@tempcntb \@ne}%
8829 \let\@hash@##%
8830 \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
8831 \def\providecommand{\@star@or@long\provide@command}
8832 \def\provide@command#1{%
8833 \begingroup
8834 \escapechar\m@ne\def\@gtempa{\string#1}%
8835 \endgroup
8836 \expandafter\@ifundefined\@gtempa
8837 {\def\reserved@a{\new@command#1}}%
8838 {\let\reserved@a\relax
8839 \def\reserved@a{\new@command\reserved@a}}%
8840 \reserved@a}%

```



```

8841 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
8842 \def\declare@robustcommand#1{%
8843   \edef\reserved@a{\string#1}%
8844   \def\reserved@b{#1}%
8845   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
8846   \edef#1{%
8847     \ifx\reserved@a\reserved@b
8848       \noexpand\x@protect
8849       \noexpand#1%
8850     \fi
8851     \noexpand\protect
8852     \expandafter\noexpand\csname
8853       \expandafter\@gobble\string#1 \endcsname
8854   }%
8855   \expandafter\new@command\csname
8856     \expandafter\@gobble\string#1 \endcsname
8857 }
8858 \def\x@protect#1{%
8859   \ifx\protect\@typeset@protect\else
8860     \@x@protect#1%
8861   \fi
8862 }
8863 \catcode`\&=\z@ % Trick to hide conditionals
8864 \def\@x@protect#1&fi#2##3{&fi\protect#1}

```

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

```

8865 \def\bbl@tempa{\csname newif\endcsname&ifin@}
8866 \catcode`\&=4
8867 \ifx\in@\@undefined
8868   \def\in@#1#2{%
8869     \def\in@##1#1##2##3\in@{%
8870       \ifx\in@##2\in@false\else\in@true\fi}%
8871     \in@#2#1\in@\in@}
8872 \else
8873   \let\bbl@tempa\@empty
8874 \fi
8875 \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).

```

8876 \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.

```

8877 \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  $\TeX$  environments.

```

8878 \ifx\@tempcnta\@undefined
8879   \csname newcount\endcsname\@tempcnta\relax
8880 \fi
8881 \ifx\@tempcntb\@undefined
8882   \csname newcount\endcsname\@tempcntb\relax
8883 \fi

```

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

```

8884 \ifx\bye\@undefined

```

```

8885 \advance\count10 by -2\relax
8886 \fi
8887 \ifx\@ifnextchar\@undefined
8888 \def\@ifnextchar#1#2#3{%
8889 \let\reserved@d=#1%
8890 \def\reserved@a{#2}\def\reserved@b{#3}%
8891 \futurelet\@let@token\@ifnch}
8892 \def\@ifnch{%
8893 \ifx\@let@token\@sptoken
8894 \let\reserved@c\@xifnch
8895 \else
8896 \ifx\@let@token\reserved@d
8897 \let\reserved@c\reserved@a
8898 \else
8899 \let\reserved@c\reserved@b
8900 \fi
8901 \fi
8902 \reserved@c}
8903 \def\:{\let\@sptoken= }\: % this makes \@sptoken a space token
8904 \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
8905 \fi
8906 \def\@testopt#1#2{%
8907 \@ifnextchar[#{1}{#1[#2]}}
8908 \def\@protected@testopt#1{%
8909 \ifx\protect\@typeset@protect
8910 \expandafter\@testopt
8911 \else
8912 \@x@protect#1%
8913 \fi}
8914 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8915 #2\relax}\fi}
8916 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8917 \else\expandafter\@gobble\fi{#1}}

```

## 15.4 Encoding related macros

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

```

8918 \def\DeclareTextCommand{%
8919 \@dec@text@cmd\providecommand
8920 }
8921 \def\ProvideTextCommand{%
8922 \@dec@text@cmd\providecommand
8923 }
8924 \def\DeclareTextSymbol#1#2#3{%
8925 \@dec@text@cmd\chardef#1{#2}#3\relax
8926 }
8927 \def\@dec@text@cmd#1#2#3{%
8928 \expandafter\def\expandafter#2%
8929 \expandafter{%
8930 \csname#3-cmd\expandafter\endcsname
8931 \expandafter#2%
8932 \csname#3\string#2\endcsname
8933 }%
8934 % \let\@ifdefinable\@rc@ifdefinable
8935 \expandafter#1\csname#3\string#2\endcsname
8936 }
8937 \def\@current@cmd#1{%
8938 \ifx\protect\@typeset@protect\else
8939 \noexpand#1\expandafter\@gobble
8940 \fi
8941 }
8942 \def\@changed@cmd#1#2{%
8943 \ifx\protect\@typeset@protect

```

```

8944 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8945 \expandafter\ifx\csname ?\string#1\endcsname\relax
8946 \expandafter\def\csname ?\string#1\endcsname{%
8947 \@changed@x@err{#1}%
8948 }%
8949 \fi
8950 \global\expandafter\let
8951 \csname\cf@encoding\string#1\expandafter\endcsname
8952 \csname ?\string#1\endcsname
8953 \fi
8954 \csname\cf@encoding\string#1%
8955 \expandafter\endcsname
8956 \else
8957 \noexpand#1%
8958 \fi
8959 }
8960 \def\@changed@x@err#1{%
8961 \errhelp{Your command will be ignored, type <return> to proceed}%
8962 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8963 \def\DeclareTextCommandDefault#1{%
8964 \DeclareTextCommand#1?%
8965 }
8966 \def\ProvideTextCommandDefault#1{%
8967 \ProvideTextCommand#1?%
8968 }
8969 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8970 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8971 \def\DeclareTextAccent#1#2#3{%
8972 \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
8973 }
8974 \def\DeclareTextCompositeCommand#1#2#3#4{%
8975 \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8976 \edef\reserved@b{\string##1}%
8977 \edef\reserved@c{%
8978 \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8979 \ifx\reserved@b\reserved@c
8980 \expandafter\expandafter\expandafter\ifx
8981 \expandafter\@car\reserved@a\relax\relax\nil
8982 \@text@composite
8983 \else
8984 \edef\reserved@b##1{%
8985 \def\expandafter\noexpand
8986 \csname#2\string#1\endcsname###1{%
8987 \noexpand\@text@composite
8988 \expandafter\noexpand\csname#2\string#1\endcsname
8989 ###1\noexpand\@empty\noexpand\@text@composite
8990 {##1}%
8991 }%
8992 }%
8993 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8994 \fi
8995 \expandafter\def\csname\expandafter\string\csname
8996 #2\endcsname\string#1-\string#3\endcsname{#4}
8997 \else
8998 \errhelp{Your command will be ignored, type <return> to proceed}%
8999 \errmessage{\string\DeclareTextCompositeCommand\space used on
9000 inappropriate command \protect#1}
9001 \fi
9002 }
9003 \def\@text@composite#1#2#3\@text@composite{%
9004 \expandafter\@text@composite@x
9005 \csname\string#1-\string#2\endcsname
9006 }

```

```

9007 \def\@text@composite@x#1#2{%
9008   \ifx#1\relax
9009     #2%
9010   \else
9011     #1%
9012   \fi
9013 }
9014 %
9015 \def\@strip@args#1:#2-#3\@strip@args{#2}
9016 \def\DeclareTextComposite#1#2#3#4{%
9017   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
9018   \bgroup
9019     \lccode`\@=#4%
9020     \lowercase{%
9021   \egroup
9022   \reserved@a @%
9023   }%
9024 }
9025 %
9026 \def\UseTextSymbol#1#2{#2}
9027 \def\UseTextAccent#1#2#3{}
9028 \def\@use@text@encoding#1{}
9029 \def\DeclareTextSymbolDefault#1#2{%
9030   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
9031 }
9032 \def\DeclareTextAccentDefault#1#2{%
9033   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
9034 }
9035 \def\cf@encoding{OT1}

```

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

```

9036 \DeclareTextAccent{\"}{OT1}{127}
9037 \DeclareTextAccent{\'}{OT1}{19}
9038 \DeclareTextAccent{\^}{OT1}{94}
9039 \DeclareTextAccent{\`}{OT1}{18}
9040 \DeclareTextAccent{\~}{OT1}{126}

```

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

```

9041 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
9042 \DeclareTextSymbol{\textquotedblright}{OT1}{`\'}
9043 \DeclareTextSymbol{\textquoteleft}{OT1}{`\'}
9044 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
9045 \DeclareTextSymbol{\i}{OT1}{16}
9046 \DeclareTextSymbol{\ss}{OT1}{25}

```

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

```

9047 \ifx\scriptsize\@undefined
9048   \let\scriptsize\sevenrm
9049 \fi

```

And a few more “dummy” definitions.

```

9050 \def\language{english}%
9051 \let\bbl@opt@shorthands\@nnil
9052 \def\bbl@ifshorthand#1#2#3{#2}%
9053 \let\bbl@language@opts\@empty
9054 \let\bbl@ensureinfo\@gobble
9055 \let\bbl@provide@locale\relax
9056 \ifx\babeloptionstrings\@undefined
9057   \let\bbl@opt@strings\@nnil
9058 \else
9059   \let\bbl@opt@strings\babeloptionstrings
9060 \fi

```

```

9061 \def\BabelStringsDefault{generic}
9062 \def\bbl@tempa{normal}
9063 \ifx\babeloptionmath\bbl@tempa
9064   \def\bbl@mathnormal{\noexpand\textormath}
9065 \fi
9066 \def\AfterBabelLanguage#1#2{}
9067 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
9068 \let\bbl@afterlang\relax
9069 \def\bbl@opt@safe{BR}
9070 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
9071 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
9072 \expandafter\newif\csname ifbbl@single\endcsname
9073 \chardef\bbl@bidimode\z@
9074 <</Emulate LaTeX>>

```

A proxy file:

```

9075 <*\plain>
9076 \input babel.def
9077 </\plain>

```

## 16 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).