

# Babel

Code

Version 3.89  
2023/05/11

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	Multiple languages . . . . .	7
3.2	The Package File ( $\LaTeX$ , <code>babel.sty</code> ) . . . . .	8
3.3	<code>base</code> . . . . .	9
3.4	<code>key=value</code> options and other general option . . . . .	10
3.5	Conditional loading of shorthands . . . . .	11
3.6	Interlude for Plain . . . . .	13
<b>4</b>	<b>Multiple languages</b>	<b>13</b>
4.1	Selecting the language . . . . .	15
4.2	Errors . . . . .	23
4.3	Hooks . . . . .	25
4.4	Setting up language files . . . . .	27
4.5	Shorthands . . . . .	29
4.6	Language attributes . . . . .	38
4.7	Support for saving macro definitions . . . . .	40
4.8	Short tags . . . . .	41
4.9	Hyphens . . . . .	41
4.10	Multiencoding strings . . . . .	43
4.11	Macros common to a number of languages . . . . .	49
4.12	Making glyphs available . . . . .	49
4.12.1	Quotation marks . . . . .	49
4.12.2	Letters . . . . .	51
4.12.3	Shorthands for quotation marks . . . . .	52
4.12.4	Umlauts and tremas . . . . .	52
4.13	Layout . . . . .	53
4.14	Load engine specific macros . . . . .	54
4.15	Creating and modifying languages . . . . .	55
<b>5</b>	<b>Adjusting the Babel bahavior</b>	<b>77</b>
5.1	Cross referencing macros . . . . .	79
5.2	Marks . . . . .	81
5.3	Preventing clashes with other packages . . . . .	82
5.3.1	<code>ifthen</code> . . . . .	82
5.3.2	<code>varioref</code> . . . . .	83
5.3.3	<code>hhline</code> . . . . .	83
5.4	Encoding and fonts . . . . .	84
5.5	Basic bidi support . . . . .	86
5.6	Local Language Configuration . . . . .	89
5.7	Language options . . . . .	89
<b>6</b>	<b>The kernel of Babel (<code>babel.def</code>, <code>common</code>)</b>	<b>92</b>
<b>7</b>	<b>Loading hyphenation patterns</b>	<b>93</b>
<b>8</b>	<b>Font handling with <code>fontspec</code></b>	<b>97</b>
<b>9</b>	<b>Hooks for XeTeX and LuaTeX</b>	<b>100</b>
9.1	XeTeX . . . . .	100
9.2	Layout . . . . .	102
9.3	8-bit TeX . . . . .	104
9.4	LuaTeX . . . . .	104
9.5	Southeast Asian scripts . . . . .	110
9.6	CJK line breaking . . . . .	112

9.7	Arabic justification . . . . .	114
9.8	Common stuff . . . . .	118
9.9	Automatic fonts and ids switching . . . . .	118
9.10	Bidi . . . . .	124
9.11	Layout . . . . .	126
9.12	Lua: transforms . . . . .	133
9.13	Lua: Auto bidi with basic and basic-r . . . . .	141
<b>10</b>	<b>Data for CJK</b>	<b>151</b>
<b>11</b>	<b>The ‘nil’ language</b>	<b>152</b>
<b>12</b>	<b>Calendars</b>	<b>153</b>
12.1	Islamic . . . . .	153
12.2	Hebrew . . . . .	154
12.3	Persian . . . . .	159
12.4	Coptic and Ethiopic . . . . .	159
12.5	Buddhist . . . . .	160
<b>13</b>	<b>Support for Plain T<sub>E</sub>X (plain.def)</b>	<b>160</b>
13.1	Not renaming hyphen.tex . . . . .	160
13.2	Emulating some L <sup>A</sup> T <sub>E</sub> X features . . . . .	161
13.3	General tools . . . . .	161
13.4	Encoding related macros . . . . .	165
<b>14</b>	<b>Acknowledgements</b>	<b>168</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 appropriated places in the source code and defined with either `<<name=value>>`, or with a series of lines between `<<*name>>` and `<</name>>`. The latter is cumulative (eg, with *More package options*). That brings a little bit of literate programming. The guards `<-name>` and `<+name>` have been redefined, too. See babel.ins for further details.

## 2 locale directory

A required component of babel is a set of ini files with basic definitions for about 250 languages. They are distributed as a separate zip file, not packed as dtx. Most of them are essentially finished (except bugs and mistakes, of course). Some of them are still incomplete (but they will be usable), and there are some omissions (eg, there are no geographic areas in Spanish). Not all include LICR variants.

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

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

## 3 Tools

```
1 <<version=3.89>>
2 <<date=2023/05/11>>
```

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

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

```
3 <<*Basic macros>> ≡
4 \bbl@trace{Basic macros}
5 \def\bbl@stripslash{\expandafter\@gobble\string}
6 \def\bbl@add#1#2{%
7   \bbl@ifunset{\bbl@stripslash#1}%
8     {\def#1{#2}}%
9     {\expandafter\def\expandafter#1\expandafter{#1#2}}
10 \def\bbl@xin@{\@expandtwoargs\in@}
11 \def\bbl@carg#1#2{\expandafter#1\csname#2\endcsname}%
12 \def\bbl@ncarg#1#2#3{\expandafter#1\expandafter#2\csname#3\endcsname}%
13 \def\bbl@ccarg#1#2#3{%
14   \expandafter#1\csname#2\expandafter\endcsname\csname#3\endcsname}%
15 \def\bbl@csarg#1#2{\expandafter#1\csname bbl@#2\endcsname}%
16 \def\bbl@cs#1{\csname bbl@#1\endcsname}
17 \def\bbl@c1#1{\csname bbl@#1\language\endcsname}
```

```

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

```

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

```

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

```

**\bbl@afterelse** Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the `\else` and `\fi` parts of an `\if`-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}}

```

**\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 `\ifcsname`, which is more efficient, and does not waste

<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.

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

```

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

```

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

```

76 \def\bbl@ifblank#1{%
77   \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
78 \long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil#4}
79 \def\bbl@ifset#1#2#3{%
80   \bbl@ifunset{#1}{#3}{\bbl@exp{\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 ckecking the replacement is really necessary or just paranoia).

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

### 3.1 Multiple languages

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

```

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

```



```

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

```

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

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

```

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

```

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

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

```

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

```

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

```

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

```

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

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

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

```

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

```

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

```

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

```

### 3.3 base

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

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

```

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

```

### 3.4 key=value options and other general option

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

```

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

```

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

```

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

```

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

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

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

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

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

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

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

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

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

### 3.5 Conditional loading of shorthands

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

435     \expandafter\@secondoftwo
436     \fi}
437 \fi
438 \endpackage
439 \let\@core\@core

```

### 3.6 Interlude for Plain

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

```

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

```

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

```

448 \endcore
449 \let\@core\@core

```

## 4 Multiple languages

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

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

```

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

```

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

```

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

```

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

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

```

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

```

```

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

```

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

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

```

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

```

```

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

```

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

```

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

```

## 4.1 Selecting the language

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

```

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

```

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

```

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

```

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

```

573 \let\xstring\string

```



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```

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

```

The unprotected part of \selectlanguage.

```

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

```

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

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

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

```

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

```

```

649 \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
650 \bbl@savelastskip
651 \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
652 \bbl@restorelastskip
653 \fi
654 \bbl@usehooks{write}}}%
655 \fi
656 \fi}
657 %
658 \let\bbl@restorelastskip\relax
659 \let\bbl@savelastskip\relax
660 %
661 \newif\ifbbl@bcpallowed
662 \bbl@bcpallowedfalse
663 \def\select@language#1{% from set@, babel@aux
664 \ifx\bbl@select@name\@empty
665 \def\bbl@select@name{select}%
666 % set hmap
667 \fi
668 \ifnum\bbl@hmapsel=\@cclv\chardef\bbl@hmapsel4\relax\fi
669 % set name
670 \edef\language#1}%
671 \bbl@fixname\language
672 % TODO. name@map must be here?
673 \bbl@provide@locale
674 \bbl@iflanguage\language{%
675 \let\bbl@select@type\z@
676 \expandafter\bbl@switch\expandafter{\language}}%
677 \def\babel@aux#1#2{%
678 \select@language{#1}%
679 \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
680 \writefile{##1}{\babel@toc{#1}{#2}\relax}}}% TODO - plain?
681 \def\babel@toc#1#2{%
682 \select@language{#1}}

```

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

The name of the language is stored in the control sequence `\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<lang>` command at definition time by expanding the `\csname` primitive. Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of `\selectlanguage`, and calling these macros.

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

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

```

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

```

```

697 \bbl@id@assign
698 % switch captions, date
699 \bbl@bsphack
700 \ifcase\bbl@select@type
701 \csname captions#1\endcsname\relax
702 \csname date#1\endcsname\relax
703 \else
704 \bbl@xin@{,captions,}{,\bbl@select@opts,}%
705 \ifin@
706 \csname captions#1\endcsname\relax
707 \fi
708 \bbl@xin@{,date,}{,\bbl@select@opts,}%
709 \ifin@ % if \foreign... within \<lang>date
710 \csname date#1\endcsname\relax
711 \fi
712 \fi
713 \bbl@esphack
714 % switch extras
715 \csname bbl@preextras@#1\endcsname
716 \bbl@usehooks{beforeextras}{}%
717 \csname extras#1\endcsname\relax
718 \bbl@usehooks{afterextras}{}%
719 % > babel-ensure
720 % > babel-sh-<short>
721 % > babel-bidi
722 % > babel-fontspec
723 \let\bbl@savedextras\empty
724 % hyphenation - case mapping
725 \ifcase\bbl@opt@hyphenmap\or
726 \def\BabelLower##1##2{\lccode##1=##2\relax}%
727 \ifnum\bbl@hymapsel>4\else
728 \csname\language\name @bbl@hyphenmap\endcsname
729 \fi
730 \chardef\bbl@opt@hyphenmap\z@
731 \else
732 \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
733 \csname\language\name @bbl@hyphenmap\endcsname
734 \fi
735 \fi
736 \let\bbl@hymapsel@cclv
737 % hyphenation - select rules
738 \ifnum\csname l@\language\endcsname=\l@unhyphenated
739 \edef\bbl@tempa{u}%
740 \else
741 \edef\bbl@tempa{\bbl@c1{lnbrk}}%
742 \fi
743 % linebreaking - handle u, e, k (v in the future)
744 \bbl@xin@{/u}{/\bbl@tempa}%
745 \ifin@\else\bbl@xin@{/e}{/\bbl@tempa}\fi % elongated forms
746 \ifin@\else\bbl@xin@{/k}{/\bbl@tempa}\fi % only kashida
747 \ifin@\else\bbl@xin@{/p}{/\bbl@tempa}\fi % padding (eg, Tibetan)
748 \ifin@\else\bbl@xin@{/v}{/\bbl@tempa}\fi % variable font
749 \ifin@
750 % unhyphenated/kashida/elongated/padding = allow stretching
751 \language\l@unhyphenated
752 \babel@savevariable\emergencystretch
753 \emergencystretch\maxdimen
754 \babel@savevariable\hbadness
755 \hbadness\@M
756 \else
757 % other = select patterns
758 \bbl@patterns{#1}%
759 \fi

```

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

`\originalTeX` The macro `\originalTeX` should be known to  $\text{\TeX}$  at this moment. As it has to be expandable we `\let` it to `\@empty` instead of `\relax`.

```

900 \ifx\originalTeX\@undefined\let\originalTeX\@empty\fi

```

Because this part of the code can be included in a format, we make sure that the macro which initializes the save mechanism, `\babel@beginsave`, is not considered to be undefined.

```

901 \ifx\babel@beginsave\@undefined\let\babel@beginsave\relax\fi

```

A few macro names are reserved for future releases of babel, which will use the concept of ‘locale’:

```

902 \providecommand\setlocale{%
903 \bbl@error
904 {Not yet available}%
905 {Find an armchair, sit down and wait}}
906 \let\uselocale\setlocale
907 \let\locale\setlocale
908 \let\selectlocale\setlocale
909 \let\textlocale\setlocale
910 \let\textlanguage\setlocale
911 \let\languagetext\setlocale

```

## 4.2 Errors

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

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

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

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

```

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

```



```

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

```

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

```

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

```

\addto It takes two arguments, a *<control sequence>* and T<sub>E</sub>X-code to be added to the *<control sequence>*.

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

```

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

```

The macro `\initiate@active@char` below takes all the necessary actions to make its argument a shorthand character. The real work is performed once for each character. But first we define a little tool.

```

987 \def\bbl@withactive#1#2{%
988   \begingroup
989     \lccode`~=#2\relax
990     \lowercase{\endgroup#1~}}

```

`\bbl@redefine` To redefine a command, we save the old meaning of the macro. Then we redefine it to call the original macro with the ‘sanitized’ argument. The reason why we do it this way is that we don’t want to redefine the  $\TeX$  macros completely in case their definitions change (they have changed in the past). A macro named `\macro` will be saved new control sequences named `\org@macro`.

```

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

```

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

```

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

```

`\bbl@redefineroobust` For commands that are redefined, but which *might* be robust we need a slightly more intelligent macro. A robust command `foo` is defined to expand to `\protect\foo_`. So it is necessary to check whether `\foo_` exists. The result is that the command that is being redefined is always robust afterwards. Therefore all we need to do now is define `\foo_`.

```

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

```

### 4.3 Hooks

Admittedly, the current implementation is a somewhat simplistic and does very little to catch errors, but it is meant for developers, after all. `\bbl@usehooks` is the commands used by `babel` to execute hooks defined for an event.

```

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

```

```

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

```

To ensure forward compatibility, arguments in hooks are set implicitly. So, if a further argument is added in the future, there is no need to change the existing code. Note events intended for hyphen.cfg are also loaded (just in case you need them for some reason).

```

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

```

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

```

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

```

```

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

```

## 4.4 Setting up language files

`\LdfInit` `\LdfInit` macro takes two arguments. The first argument is the name of the language that will be defined in the language definition file; the second argument is either a control sequence or a string from which a control sequence should be constructed. The existence of the control sequence indicates that the file has been processed before.

At the start of processing a language definition file we always check the category code of the `@`-sign. We make sure that it is a ‘letter’ during the processing of the file. We also save its name as the last called option, even if not loaded.

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

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

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

\endinput  
 When #2 was *not* a control sequence we construct one and compare it with \relax.  
 Finally we check \originalTeX.

```

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

```

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

```

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

```

\ldf@finish This macro takes one argument. It is the name of the language that was defined in the language definition file.

We load the local configuration file if one is present, we set the main language (taking into account that the argument might be a control sequence that needs to be expanded) and reset the category code of the @-sign.

```

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

```

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

```

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

```

`\main@language` This command should be used in the various language definition files. It stores its argument in `\bbl@main@language`; to be used to switch to the correct language at the beginning of the document.

```

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

```

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

```

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

```

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

```

1179 \def\select@language@x#1{%
1180   \ifcase\bbl@select@type
1181     \bbl@ifsamestring\languagename{#1}{\select@language{#1}}%
1182   \else
1183     \select@language{#1}%
1184   \fi}

```

## 4.5 Shorthands

`\bbl@add@special` The macro `\bbl@add@special` is used to add a new character (or single character control sequence) to the macro `\dospecials` (and `\@sanitize` if  $\TeX$  is used). It is used only at one place, namely when `\initiate@active@char` is called (which is ignored if the char has been made active before). Because `\@sanitize` can be undefined, we put the definition inside a conditional. Items are added to the lists without checking its existence or the original catcode. It does not hurt, but should be fixed. It's already done with `\nfss@catcodes`, added in 3.10.

```

1185 \bbl@trace{Shorhands}
1186 \def\bbl@add@special#1{% 1:a macro like "\, \?, etc.
1187   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1188   \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\@makeother#1}}%
1189   \ifx\nfss@catcodes\undefined\else % TODO - same for above
1190     \begingroup
1191       \catcode`#1\active
1192       \nfss@catcodes
1193       \ifnum\catcode`#1=\active
1194         \endgroup
1195       \bbl@add\nfss@catcodes{\@makeother#1}%
1196     \else

```

```

1197     \endgroup
1198     \fi
1199   \fi}

```

`\bbl@remove@special` The companion of the former macro is `\bbl@remove@special`. It removes a character from the set macros `\dospecials` and `\@sanitize`, but it is not used at all in the babel core.

```

1200 \def\bbl@remove@special#1{%
1201   \begingroup
1202     \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1203       \else\noexpand##1\noexpand##2\fi}%
1204     \def\do{\x\do}%
1205     \def\@makeother{\x\@makeother}%
1206   \edef\x{\endgroup
1207     \def\noexpand\dospecials{\dospecials}%
1208     \expandafter\ifx\csname @sanitize\endcsname\relax\else
1209       \def\noexpand\@sanitize{\@sanitize}%
1210     \fi}%
1211   \x}

```

`\initiate@active@char` A language definition file can call this macro to make a character active. This macro takes one argument, the character that is to be made active. When the character was already active this macro does nothing. Otherwise, this macro defines the control sequence `\normal@char<char>` to expand to the character in its ‘normal state’ and it defines the active character to expand to `\normal@char<char>` by default (`<char>` being the character to be made active). Later its definition can be changed to expand to `\active@char<char>` by calling `\bbl@activate{<char>}`. For example, to make the double quote character active one could have `\initiate@active@char{"}` in a language definition file. This defines `"` as `\active@prefix "\active@char` (where the first `"` is the character with its original catcode, when the shorthand is created, and `\active@char` is a single token). In protected contexts, it expands to `\protect "` or `\noexpand "` (ie, with the original `"`); otherwise `\active@char` is executed. This macro in turn expands to `\normal@char` in “safe” contexts (eg, `\label`), but `\user@active` in normal “unsafe” ones. The latter search a definition in the user, language and system levels, in this order, but if none is found, `\normal@char` is used. However, a deactivated shorthand (with `\bbl@deactivate` is defined as `\active@prefix "\normal@char`.

The following macro is used to define shorthands in the three levels. It takes 4 arguments: the (string’ed) character, `<level>@group`, `<level>@active` and `<next-level>@active` (except in system).

```

1212 \def\bbl@active@def#1#2#3#4{%
1213   \@namedef{#3#1}{%
1214     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1215       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1216     \else
1217       \bbl@afterfi\csname#2@sh@#1\endcsname
1218     \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.

```

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

```

1225 \def\initiate@active@char#1{%
1226   \bbl@ifunset{active@char\string#1}%
1227   {\bbl@withactive
1228     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1229   {}}

```

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).

```

1230 \def\@initiate@active@char#1#2#3{%
1231   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1232   \ifx#1\@undefined
1233     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1234   \else
1235     \bbl@csarg\let{oridef@#2}#1%
1236     \bbl@csarg\edef{oridef@#2}{%
1237       \let\noexpand#1%
1238       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1239   \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*").

```

1240   \ifx#1#3\relax
1241     \expandafter\let\csname normal@char#2\endcsname#3%
1242   \else
1243     \bbl@info{Making #2 an active character}%
1244     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1245     \@namedef{normal@char#2}{%
1246       \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
1247   \else
1248     \@namedef{normal@char#2}{#3}%
1249   \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).

```

1250   \bbl@restoreactive{#2}%
1251   \AtBeginDocument{%
1252     \catcode`#2\active
1253     \if@filesw
1254       \immediate\write\@mainaux{\catcode`\string#2\active}%
1255     \fi}%
1256   \expandafter\bbl@add@special\csname#2\endcsname
1257   \catcode`#2\active
1258 \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⟩`).

```

1259   \let\bbl@tempa\@firstoftwo
1260   \if\string^#2%
1261     \def\bbl@tempa{\noexpand\textormath}%
1262   \else
1263     \ifx\bbl@mathnormal\@undefined\else
1264       \let\bbl@tempa\bbl@mathnormal
1265     \fi
1266   \fi
1267   \expandafter\edef\csname active@char#2\endcsname{%
1268     \bbl@tempa
1269     {\noexpand\if@safe@actives
1270      \noexpand\expandafter

```



```

1271      \expandafter\noexpand\csname normal@char#2\endcsname
1272      \noexpand\else
1273      \noexpand\expandafter
1274      \expandafter\noexpand\csname bbl@doactive#2\endcsname
1275      \noexpand\fi}%
1276      {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1277      \bbl@csarg\edef{doactive#2}{%
1278      \expandafter\noexpand\csname user@active#2\endcsname}%

```

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

`\active@prefix <char> \normal@char <char>`

(where `\active@char <char>` is *one* control sequence!).

```

1279      \bbl@csarg\edef{active@#2}{%
1280      \noexpand\active@prefix\noexpand#1%
1281      \expandafter\noexpand\csname active@char#2\endcsname}%
1282      \bbl@csarg\edef{normal@#2}{%
1283      \noexpand\active@prefix\noexpand#1%
1284      \expandafter\noexpand\csname normal@char#2\endcsname}%
1285      \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.

```

1286      \bbl@active@def#2\user@group{user@active}{language@active}%
1287      \bbl@active@def#2\language@group{language@active}{system@active}%
1288      \bbl@active@def#2\system@group{system@active}{normal@char}%

```

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

```

1289      \expandafter\edef\csname\user@group @sh#2@@\endcsname
1290      {\expandafter\noexpand\csname normal@char#2\endcsname}%
1291      \expandafter\edef\csname\user@group @sh#2@string\protect@\endcsname
1292      {\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.

```

1293      \if\string'#2%
1294      \let\prim@s\bbl@prim@s
1295      \let\active@math@prime#1%
1296      \fi
1297      \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}

```

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

```

1298      <<More package options>> ≡
1299      \DeclareOption{math=active}{}
1300      \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1301      <</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*.

```

1302      \@ifpackagewith{babel}{KeepShorthandsActive}%
1303      {\let\bbl@restoreactive@gobble}%
1304      {\def\bbl@restoreactive#1{%
1305      \bbl@exp{%
1306      \\AfterBabelLanguage\\CurrentOption

```

```

1307      {\catcode`#1=\the\catcode`#1\relax}%
1308      \\AtEndOfPackage
1309      {\catcode`#1=\the\catcode`#1\relax}}}%
1310      \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.

```

1311 \def\bbl@sh@select#1#2{%
1312   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1313     \bbl@afterelse\bbl@scndcs
1314   \else
1315     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1316   \fi}

```

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

```

1317 \begingroup
1318 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct? Only Plain?
1319 {\gdef\active@prefix#1{%
1320   \ifx\protect\@typeset@protect
1321   \else
1322     \ifx\protect\@unexpandable@protect
1323       \noexpand#1%
1324     \else
1325       \protect#1%
1326     \fi
1327   \expandafter\@gobble
1328   \fi}}
1329 {\gdef\active@prefix#1{%
1330   \ifincsname
1331     \string#1%
1332     \expandafter\@gobble
1333   \else
1334     \ifx\protect\@typeset@protect
1335     \else
1336       \ifx\protect\@unexpandable@protect
1337         \noexpand#1%
1338       \else
1339         \protect#1%
1340       \fi
1341       \expandafter\expandafter\expandafter\@gobble
1342     \fi
1343   \fi}}
1344 \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 `"13"13` becomes `"12"12` in an `\edef` (in other words, shorthands are `\string'ed`). This contrasts with `\protected@edef`, where catcodes are always left unchanged. Once converted, they can be used safely even after this expansion mode is deactivated (with `\@safe@activefalse`).

```

1345 \newif\if@safe@actives
1346 \@safe@activefalse

```

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

```
1347 \def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}
```

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

```
1348 \chardef\bbl@activated\z@
1349 \def\bbl@activate#1{%
1350   \chardef\bbl@activated\@ne
1351   \bbl@withactive{\expandafter\let\expandafter}#1%
1352   \csname bbl@active@\string#1\endcsname}
1353 \def\bbl@deactivate#1{%
1354   \chardef\bbl@activated\tw@
1355   \bbl@withactive{\expandafter\let\expandafter}#1%
1356   \csname bbl@normal@\string#1\endcsname}
```

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

```
\bbl@scndcs
1357 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1358 \def\bbl@scndcs#1#2{\csname#2\endcsname}
```

`\declare@shorthand` The command `\declare@shorthand` is used to declare a shorthand on a certain level. It takes three arguments:

1. a name for the collection of shorthands, i.e. ‘system’, or ‘dutch’;
2. the character (sequence) that makes up the shorthand, i.e. `~` or `"a`;
3. the code to be executed when the shorthand is encountered.

The auxiliary macro `\babel@texpdf` improves the interoperativity with `hyperref` and takes 4 arguments: (1) The  $\TeX$  code in text mode, (2) the string for `hyperref`, (3) the  $\TeX$  code in math mode, and (4), which is currently ignored, but it’s meant for a string in math mode, like a minus sign instead of an hyphen (currently `hyperref` doesn’t discriminate the mode). This macro may be used in `ldf` files.

```
1359 \def\babel@texpdf#1#2#3#4{%
1360   \ifx\texorpdfstring\@undefined
1361     \textormath{#1}{#3}%
1362   \else
1363     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1364     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1365   \fi}
1366 %
1367 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1368 \def\@decl@short#1#2#3\@nil#4{%
1369   \def\bbl@tempa{#3}%
1370   \ifx\bbl@tempa\@empty
1371     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1372     \bbl@ifunset{#1@sh@\string#2@}{}%
1373     {\def\bbl@tempa{#4}%
1374      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1375      \else
1376        \bbl@info
1377        {Redefining #1 shorthand \string#2\\%
1378         in language \CurrentOption}%
1379      \fi}%
1380     \@namedef{#1@sh@\string#2@}{#4}%
1381   \else
1382     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1383     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1384     {\def\bbl@tempa{#4}%
1385      \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1386      \else
1387        \bbl@info
```

```

1388      {Redefining #1 shorthand \string#2\string#3\\%
1389      in language \CurrentOption}%
1390    \fi}%
1391    \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1392  \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.

```

1393 \def\textormath{%
1394   \ifmmode
1395     \expandafter\@secondoftwo
1396   \else
1397     \expandafter\@firstoftwo
1398   \fi}

```

`\user@group` The current concept of ‘shorthands’ supports three levels or groups of shorthands. For each level the `\language@group` 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’.

```

1399 \def\user@group{user}
1400 \def\language@group{english} % TODO. I don't like defaults
1401 \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.

```

1402 \def\useshorthands{%
1403   \@ifstar\bb1@usesh@s{\bb1@usesh@x{}}
1404 \def\bb1@usesh@s#1{%
1405   \bb1@usesh@x
1406   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb1@activate{#1}}}%
1407   {#1}}
1408 \def\bb1@usesh@x#1#2{%
1409   \bb1@ifshorthand{#2}%
1410   {\def\user@group{user}%
1411    \initiate@active@char{#2}%
1412    #1%
1413    \bb1@activate{#2}}%
1414   {\bb1@error
1415    {I can't declare a shorthand turned off (\string#2)}
1416    {Sorry, but you can't use shorthands which have been\\%
1417     turned off in the package options}}}

```

`\defineshorthand` Currently we only support two groups of user level shorthands, named internally `user` and `user<lang>` (language-dependent user shorthands). By default, only the first one is taken into account, but if the former is also used (in the optional argument of `\defineshorthand`) a new level is inserted for it (`user@generic`, done by `\bb1@set@user@generic`); we make also sure `{}` and `\protect` are taken into account in this new top level.

```

1418 \def\user@language@group{user@\language@group}
1419 \def\bb1@set@user@generic#1#2{%
1420   \bb1@ifunset{user@generic@active#1}%
1421   {\bb1@active@def#1\user@language@group{user@active}{user@generic@active}%
1422    \bb1@active@def#1\user@group{user@generic@active}{language@active}%
1423    \expandafter\edef\csname#2@sh@#1@\endcsname{%
1424      \expandafter\noexpand\csname normal@char#1\endcsname}%
1425    \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1426      \expandafter\noexpand\csname user@active#1\endcsname}}%
1427   \@empty}
1428 \newcommand\defineshorthand[3][user]{%
1429   \edef\bb1@tempa{\zap@space#1 \@empty}%
1430   \bb1@for\bb1@tempb\bb1@tempa{%
1431     \if*\expandafter\@car\bb1@tempb\@nil
1432     \edef\bb1@tempb{user@\expandafter\@gobble\bb1@tempb}%

```

```

1433 \expandtwoargs
1434 \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1435 \fi
1436 \declare@shorthand{\bbl@tempb}{#2}{#3}}

```

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

```

1437 \def\languageshorthands#1{\def\language@group{#1}}

```

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

```

1438 \def\aliasshorthand#1#2{%
1439 \bbl@ifshorthand{#2}%
1440 {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1441 \ifx\document\@notprerr
1442 \@notshorthand{#2}%
1443 \else
1444 \initiate@active@char{#2}%
1445 \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1446 \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1447 \bbl@activate{#2}%
1448 \fi
1449 \fi}%
1450 {\bbl@error
1451 {Cannot declare a shorthand turned off (\string#2)}
1452 {Sorry, but you cannot use shorthands which have been\\%
1453 turned off in the package options}}}

```

`\@notshorthand`

```

1454 \def\@notshorthand#1{%
1455 \bbl@error{%
1456 The character '\string #1' should be made a shorthand character;\\%
1457 add the command \string\useshorthands\string{#1\string} to
1458 the preamble.\\%
1459 I will ignore your instruction}%
1460 {You may proceed, but expect unexpected results}}

```

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

```

1461 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1462 \DeclareRobustCommand*\shorthandoff{%
1463 \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1464 \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.

```

1465 \def\bbl@switch@sh#1#2{%
1466 \ifx#2\@nnil\else
1467 \bbl@ifunset{\bbl@active@\string#2}%
1468 {\bbl@error
1469 {I can't switch '\string#2' on or off--not a shorthand}%
1470 {This character is not a shorthand. Maybe you made\\%
1471 a typing mistake? I will ignore your instruction.}}%
1472 {\ifcase#1% off, on, off*
1473 \catcode`#212\relax

```

```

1474 \or
1475 \catcode`#2\active
1476 \bbl@ifunset{bbl@shdef@\string#2}%
1477 {}%
1478 {\bbl@withactive{\expandafter\let\expandafter}#2%
1479 \csname bbl@shdef@\string#2\endcsname
1480 \bbl@csarg\let{shdef@\string#2}\relax}%
1481 \ifcase\bbl@activated\or
1482 \bbl@activate{#2}%
1483 \else
1484 \bbl@deactivate{#2}%
1485 \fi
1486 \or
1487 \bbl@ifunset{bbl@shdef@\string#2}%
1488 {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1489 {}%
1490 \csname bbl@oricat@\string#2\endcsname
1491 \csname bbl@oridef@\string#2\endcsname
1492 \fi}%
1493 \bbl@afterfi\bbl@switch@sh#1%
1494 \fi}

```

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

```

1495 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1496 \def\bbl@putsh#1{%
1497 \bbl@ifunset{bbl@active@\string#1}%
1498 {\bbl@putsh@i#1\@empty\@nnil}%
1499 {\csname bbl@active@\string#1\endcsname}}
1500 \def\bbl@putsh@i#1#2\@nnil{%
1501 \csname\language@group @sh@\string#1@%
1502 \ifx\@empty#2\else\string#2\fi\endcsname}
1503 %
1504 \ifx\bbl@opt@shorthands\@nnil\else
1505 \let\bbl@s@initiate@active@char\initiate@active@char
1506 \def\initiate@active@char#1{%
1507 \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1508 \let\bbl@s@switch@sh\bbl@switch@sh
1509 \def\bbl@switch@sh#1#2{%
1510 \ifx#2\@nnil\else
1511 \bbl@afterfi
1512 \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1513 \fi}
1514 \let\bbl@s@activate\bbl@activate
1515 \def\bbl@activate#1{%
1516 \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1517 \let\bbl@s@deactivate\bbl@deactivate
1518 \def\bbl@deactivate#1{%
1519 \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1520 \fi

```

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

```

1521 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{bbl@active@\string#1}{#3}{#2}}

```

\bbl@prim@s One of the internal macros that are involved in substituting \prime for each right quote in  
\bbl@pr@m@s mathmode is \prim@s. This checks if the next character is a right quote. When the right quote is active, the definition of this macro needs to be adapted to look also for an active right quote; the hat could be active, too.

```

1522 \def\bbl@prim@s{%
1523 \prime\futurelet\@let@token\bbl@pr@m@s}
1524 \def\bbl@if@primes#1#2{%
1525 \ifx#1\@let@token
1526 \expandafter\@firstoftwo

```

```

1527 \else\ifx#2\@let@token
1528 \bbl@afterelse\expandafter\@firstoftwo
1529 \else
1530 \bbl@afterfi\expandafter\@secondoftwo
1531 \fi\fi}
1532 \begingroup
1533 \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
1534 \catcode`\'=12 \catcode`\\"=\active \lccode`\\"=\'
1535 \lowercase{%
1536 \gdef\bbl@pr@m@s{%
1537 \bbl@if@primes" '%
1538 \pr@@@s
1539 {\bbl@if@primes*^ \pr@@@t\egroup}}
1540 \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).

```

1541 \initiate@active@char{~}
1542 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1543 \bbl@activate{~}

```

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

```

1544 \expandafter\def\csname OT1dqpos\endcsname{127}
1545 \expandafter\def\csname T1dqpos\endcsname{4}

```

When the macro \f@encoding is undefined (as it is in plain T<sub>E</sub>X) we define it here to expand to OT1

```

1546 \ifx\f@encoding\@undefined
1547 \def\f@encoding{OT1}
1548 \fi

```

## 4.6 Language attributes

Language attributes provide a means to give the user control over which features of the language definition files he wants to enable.

\languageattribute The macro \languageattribute checks whether its arguments are valid and then activates the selected language attribute. First check whether the language is known, and then process each attribute in the list.

```

1549 \bbl@trace{Language attributes}
1550 \newcommand\languageattribute[2]{%
1551 \def\bbl@tempc{#1}%
1552 \bbl@fixname\bbl@tempc
1553 \bbl@iflanguage\bbl@tempc{%
1554 \bbl@vforeach{#2}{%

```

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

```

1555 \ifx\bbl@known@attribs\@undefined
1556 \in@false
1557 \else
1558 \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
1559 \fi
1560 \ifin@
1561 \bbl@warning{%
1562 You have more than once selected the attribute '##1'\%
1563 for language #1. Reported}%
1564 \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.

```

1565 \bbl@exp{%
1566   \bbl@add@list\bbl@known@attribs{\bbl@tempc-##1}}%
1567 \edef\bbl@tempa{\bbl@tempc-##1}%
1568 \expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes%
1569 {\csname\bbl@tempc @attr##1\endcsname}%
1570 {\@attrerr{\bbl@tempc}{##1}}%
1571 \fi}}
1572 \@onlypreamble\languageattribute

```

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

```

1573 \newcommand*{\@attrerr}[2]{%
1574   \bbl@error
1575   {The attribute #2 is unknown for language #1.}%
1576   {Your command will be ignored, type <return> to proceed}}

```

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

```

1577 \def\bbl@declare@ttribute#1#2#3{%
1578   \bbl@xin@{, #2, }{\BabelModifiers,}%
1579   \ifin@
1580     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1581   \fi
1582   \bbl@add@list\bbl@attributes{#1-#2}%
1583   \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.

```

1584 \def\bbl@ifattributeset#1#2#3#4{%
1585   \ifx\bbl@known@attribs\undefined
1586     \in@false
1587   \else
1588     \bbl@xin@{, #1-#2, }{\bbl@known@attribs,}%
1589   \fi
1590   \ifin@
1591     \bbl@afterelse#3%
1592   \else
1593     \bbl@afterfi#4%
1594   \fi}

```

**\bbl@ifknown@trib** 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.

```

1595 \def\bbl@ifknown@trib#1#2{%
1596   \let\bbl@tempa\@secondoftwo
1597   \bbl@loopx\bbl@tempb{#2}{%
1598     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{, #1,}%
1599     \ifin@
1600       \let\bbl@tempa\@firstoftwo
1601     \else
1602       \fi}%
1603   \bbl@tempa}

```



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

```
1604 \def\bbl@clear@ttribs{%
1605   \ifx\bbl@attributes\@undefined\else
1606     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1607       \expandafter\bbl@clear@ttrib\bbl@tempa.}%
1608     \let\bbl@attributes\@undefined
1609   \fi}
1610 \def\bbl@clear@ttrib#1-#2.{%
1611   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1612 \AtBeginDocument{\bbl@clear@ttribs}
```

## 4.7 Support for saving macro definitions

To save the meaning of control sequences using `\babel@save`, we use temporary control sequences. To save hash table entries for these control sequences, we don't use the name of the control sequence to be saved to construct the temporary name. Instead we simply use the value of a counter, which is reset to zero each time we begin to save new values. This works well because we release the saved meanings before we begin to save a new set of control sequence meanings (see `\selectlanguage` and `\originalTeX`). Note undefined macros are not undefined any more when saved – they are `\relax`'ed.

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

`\babel@beginsave`

```
1613 \bbl@trace{Macros for saving definitions}
1614 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1615 \newcount\babel@savecnt
1616 \babel@beginsave
```

`\babel@save` 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.

```
1617 \def\babel@save#1{%
1618   \def\bbl@tempa{,{#1,}}% Clumsy, for Plain
1619   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1620     \expandafter\expandafter,\bbl@savedextras,}%
1621   \expandafter\in@\bbl@tempa
1622   \ifin@\else
1623     \bbl@add\bbl@savedextras{,{#1,}}%
1624     \bbl@carg\let\babel@number\babel@savecnt\#1\relax
1625     \toks@\expandafter{\originalTeX\let#1=}%
1626     \bbl@exp{%
1627       \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1628     \advance\babel@savecnt\@ne
1629   \fi}
1630 \def\babel@savevariable#1{%
1631   \toks@\expandafter{\originalTeX #1=}%
1632   \bbl@exp{\def\\originalTeX{\the\toks@the#1\relax}}}
```

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

```
1633 \def\bbl@frenchspacing{%
1634   \ifnum\the\scode`\.=\@m
```

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

```

1635 \let\bbl@nonfrenchspacing\relax
1636 \else
1637 \frenchspacing
1638 \let\bbl@nonfrenchspacing\nonfrenchspacing
1639 \fi}
1640 \let\bbl@nonfrenchspacing\nonfrenchspacing
1641 \let\bbl@elt\relax
1642 \edef\bbl@fs@chars{%
1643 \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1644 \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1645 \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1646 \def\bbl@pre@fs{%
1647 \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1648 \edef\bbl@save@sfcodes{\bbl@fs@chars}}%
1649 \def\bbl@post@fs{%
1650 \bbl@save@sfcodes
1651 \edef\bbl@tempa{\bbl@cl{frspc}}%
1652 \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1653 \if u\bbl@tempa % do nothing
1654 \else\if n\bbl@tempa % non french
1655 \def\bbl@elt##1##2##3{%
1656 \ifnum\sfcode`##1=##2\relax
1657 \babel@savevariable{\sfcode`##1}%
1658 \sfcode`##1=##3\relax
1659 \fi}%
1660 \bbl@fs@chars
1661 \else\if y\bbl@tempa % french
1662 \def\bbl@elt##1##2##3{%
1663 \ifnum\sfcode`##1=##3\relax
1664 \babel@savevariable{\sfcode`##1}%
1665 \sfcode`##1=##2\relax
1666 \fi}%
1667 \bbl@fs@chars
1668 \fi\fi\fi}

```

## 4.8 Short tags

`\babeltags` This macro is straightforward. After zapping spaces, we loop over the list and define the macros `\text<tag>` and `\<tag>`. Definitions are first expanded so that they don't contain `\csname` but the actual macro.

```

1669 \bbl@trace{Short tags}
1670 \def\babeltags#1{%
1671 \edef\bbl@tempa{\zap@space#1 \@empty}%
1672 \def\bbl@tempb##1=##2\@{#}%
1673 \edef\bbl@tempc{%
1674 \noexpand\newcommand
1675 \expandafter\noexpand\csname ##1\endcsname{%
1676 \noexpand\protect
1677 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1678 \noexpand\newcommand
1679 \expandafter\noexpand\csname text##1\endcsname{%
1680 \noexpand\foreignlanguage{##2}}}
1681 \bbl@tempc}%
1682 \bbl@for\bbl@tempa\bbl@tempa{%
1683 \expandafter\bbl@tempb\bbl@tempa\@{}}

```

## 4.9 Hyphens

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

```

1684 \bbl@trace{Hyphens}

```

```

1685 \@onlypreamble\babelhyphenation
1686 \AtEndOfPackage{%
1687   \newcommand\babelhyphenation[2][\@empty]{%
1688     \ifx\bbbl@hyphenation@relax
1689       \let\bbbl@hyphenation@\@empty
1690     \fi
1691     \ifx\bbbl@hyphlist\@empty\else
1692       \bbbl@warning{%
1693         You must not intermingle \string\selectlanguage\space and\%
1694         \string\babelhyphenation\space or some exceptions will not\%
1695         be taken into account. Reported}%
1696     \fi
1697     \ifx\@empty#1%
1698       \protected@edef\bbbl@hyphenation@\bbbl@hyphenation@space#2}%
1699     \else
1700       \bbbl@vforeach{#1}{%
1701         \def\bbbl@tempa{##1}%
1702         \bbbl@fixname\bbbl@tempa
1703         \bbbl@iflanguage\bbbl@tempa{%
1704           \bbbl@csarg\protected@edef{hyphenation@\bbbl@tempa}{%
1705             \bbbl@ifunset{bbbl@hyphenation@\bbbl@tempa}%
1706               {}%
1707               {\csname bbbl@hyphenation@\bbbl@tempa\endcsname\space}%
1708             #2}}}%
1709       \fi}}

```

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

```

1710 \def\bbbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1711 \def\bbbl@t@one{T1}
1712 \def\allowhyphens{\ifx\cf@encoding\bbbl@t@one\else\bbbl@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`.

```

1713 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1714 \def\babelhyphen{\active@prefix\babelhyphen\bbbl@hyphen}
1715 \def\bbbl@hyphen{%
1716   \@ifstar{\bbbl@hyphen@i @}{\bbbl@hyphen@i \@empty}}
1717 \def\bbbl@hyphen@i#1#2{%
1718   \bbbl@ifunset{bbbl@hy@#1#2\@empty}%
1719     {\csname bbbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1720     {\csname bbbl@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.

```

1721 \def\bbbl@usehyphen#1{%
1722   \leavevmode
1723   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1724   \nobreak\hskip\z@skip}
1725 \def\bbbl@@usehyphen#1{%
1726   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1727 \def\bbbl@hyphenchar{%
1728   \ifnum\hyphenchar\font=\m@ne

```

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

```

1729 \babeInnullhyphen
1730 \else
1731 \char\hyphenchar\font
1732 \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.

```

1733 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1734 \def\bbl@hy@@soft{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1735 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1736 \def\bbl@hy@@hard{\bbl@@usehyphen\bbl@hyphenchar}
1737 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}{}}
1738 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1739 \def\bbl@hy@repeat{%
1740   \bbl@usehyphen{%
1741     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1742 \def\bbl@hy@@repeat{%
1743   \bbl@@usehyphen{%
1744     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1745 \def\bbl@hy@empty{\hskip\z@skip}
1746 \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.

```

1747 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#1}\bbl@allowhyphens}

```

## 4.10 Multiencoding strings

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

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

```

1748 \bbl@trace{Multiencoding strings}
1749 \def\bbl@tglobal#1{\global\let#1#1}

```

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

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

and starts over (and similarly when lowercasing).

```

1750 \@ifpackagewith{babel}{nocase}%
1751   {\let\bbl@patchuclc\relax}%
1752   {\def\bbl@patchuclc{% TODO. Delete. Doesn't work any more.
1753     \global\let\bbl@patchuclc\relax
1754     \gaddto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1755     \gdef\bbl@uclc##1{%
1756       \let\bbl@encoded\bbl@encoded@uclc
1757       \bbl@ifunset{\language @bbl@uclc}% and resumes it
1758       {##1}%
1759       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1760         \csname\language @bbl@uclc\endcsname}%
1761       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}}%
1762   \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
1763   \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}

```

```

1764 <<*More package options>> ≡
1765 \DeclareOption{nocase}{}
1766 <</More package options>>

```

The following package options control the behavior of \SetString.

```

1767 <<*More package options>> ≡
1768 \let\bbl@opt@strings\@nnil % accept strings=value
1769 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1770 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1771 \def\BabelStringsDefault{generic}
1772 <</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.

```

1773 \@onlypreamble\StartBabelCommands
1774 \def\StartBabelCommands{%
1775   \begingroup
1776   \@tempcnta="7F
1777   \def\bbl@tempa{%
1778     \ifnum\@tempcnta>"FF\else
1779       \catcode\@tempcnta=11
1780       \advance\@tempcnta\@ne
1781       \expandafter\bbl@tempa
1782     \fi}%
1783   \bbl@tempa
1784   <<Macros local to BabelCommands>>
1785   \def\bbl@provstring##1##2{%
1786     \providecommand##1{##2}%
1787     \bbl@tglobal##1}%
1788   \global\let\bbl@scafter\@empty
1789   \let\StartBabelCommands\bbl@startcmds
1790   \ifx\BabelLanguages\relax
1791     \let\BabelLanguages\CurrentOption
1792   \fi
1793   \begingroup
1794   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1795   \StartBabelCommands}
1796 \def\bbl@startcmds{%
1797   \ifx\bbl@screset\@nnil\else
1798     \bbl@usehooks{stopcommands}{}%
1799   \fi
1800   \endgroup
1801   \begingroup
1802   \@ifstar
1803     {\ifx\bbl@opt@strings\@nnil
1804       \let\bbl@opt@strings\BabelStringsDefault
1805     \fi
1806     \bbl@startcmds@i}%
1807   \bbl@startcmds@i}
1808 \def\bbl@startcmds@i#1#2{%
1809   \edef\bbl@L{\zap@space#1 \@empty}%
1810   \edef\bbl@G{\zap@space#2 \@empty}%
1811   \bbl@startcmds@ii}
1812 \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.

```

1813 \newcommand\bb1@startcmds@ii[1][\@empty]{%
1814   \let\SetString\@gobbletwo
1815   \let\bb1@stringdef\@gobbletwo
1816   \let\AfterBabelCommands\@gobble
1817   \ifx\@empty#1%
1818     \def\bb1@sc@label{generic}%
1819     \def\bb1@encstring##1##2{%
1820       \ProvideTextCommandDefault##1{##2}%
1821       \bb1@toglobal##1%
1822       \expandafter\bb1@toglobal\csname\string?\string##1\endcsname}%
1823     \let\bb1@sctest\in@true
1824   \else
1825     \let\bb1@sc@charset\space % <- zapped below
1826     \let\bb1@sc@fontenc\space % <- " "
1827     \def\bb1@tempa##1=##2\@nil{%
1828       \bb1@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1829     \bb1@foreach{label=#1}{\bb1@tempa##1\@nil}%
1830     \def\bb1@tempa##1 ##2{% space -> comma
1831       ##1%
1832       \ifx\@empty##2\else\ifx,##1,\else,\fi\bb1@afterfi\bb1@tempa##2\fi}%
1833     \edef\bb1@sc@fontenc{\expandafter\bb1@tempa\bb1@sc@fontenc\@empty}%
1834     \edef\bb1@sc@label{\expandafter\zap@space\bb1@sc@label\@empty}%
1835     \edef\bb1@sc@charset{\expandafter\zap@space\bb1@sc@charset\@empty}%
1836     \def\bb1@encstring##1##2{%
1837       \bb1@foreach\bb1@sc@fontenc{%
1838         \bb1@ifunset{T@###1}%
1839         {}%
1840         {\ProvideTextCommand##1{###1}{##2}%
1841          \bb1@toglobal##1%
1842          \expandafter
1843          \bb1@toglobal\csname####1\string##1\endcsname}}}%
1844     \def\bb1@sctest{%
1845       \bb1@xin@{\bb1@opt@strings,}{,\bb1@sc@label,\bb1@sc@fontenc,}}%
1846   \fi
1847   \ifx\bb1@opt@strings\@nnil      % ie, no strings key -> defaults
1848   \else\ifx\bb1@opt@strings\relax % ie, strings=encoded
1849     \let\AfterBabelCommands\bb1@aftercmds
1850     \let\SetString\bb1@setstring
1851     \let\bb1@stringdef\bb1@encstring
1852   \else % ie, strings=value
1853     \bb1@sctest
1854   \fin@
1855     \let\AfterBabelCommands\bb1@aftercmds
1856     \let\SetString\bb1@setstring
1857     \let\bb1@stringdef\bb1@provstring
1858   \fi\fi\fi
1859   \bb1@scswitch
1860   \ifx\bb1@G\@empty
1861     \def\SetString##1##2{%
1862       \bb1@error{Missing group for string \string##1}%
1863       {You must assign strings to some category, typically\\
1864        captions or extras, but you set none}}%
1865   \fi
1866   \ifx\@empty#1%
1867     \bb1@usehooks{defaultcommands}{}%
1868   \else
1869     \@expandtwoargs
1870     \bb1@usehooks{encodedcommands}{\bb1@sc@charset}{\bb1@sc@fontenc}}%
1871   \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).

```

1872 \def\bbl@forlang#1#2{%
1873   \bbl@for#1\bbl@L{%
1874     \bbl@xin@{,#1,}{, \BabelLanguages,}%
1875     \ifin@#2\relax\fi}}
1876 \def\bbl@scswitch{%
1877   \bbl@forlang\bbl@tempa{%
1878     \ifx\bbl@G\@empty\else
1879       \ifx\SetString\@gobbletwo\else
1880         \edef\bbl@GL{\bbl@G\bbl@tempa}%
1881         \bbl@xin@{,\bbl@GL,}{, \bbl@screset,}%
1882         \ifin@\else
1883           \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1884           \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1885         \fi
1886       \fi
1887     \fi}}
1888 \AtEndOfPackage{%
1889   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}}{#2}}}%
1890   \let\bbl@scswitch\relax}
1891 \@onlypreamble\EndBabelCommands
1892 \def\EndBabelCommands{%
1893   \bbl@usehooks{stopcommands}}}%
1894   \endgroup
1895   \endgroup
1896   \bbl@scafter}
1897 \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.

```

1898 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1899   \bbl@forlang\bbl@tempa{%
1900     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1901     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1902     {\bbl@exp{%
1903       \global\bbbl@add\<\bbl@G\bbl@tempa>\bbbl@scset\#1\<\bbl@LC>}}}%
1904     {}%
1905     \def\BabelString{#2}%
1906     \bbl@usehooks{stringprocess}}}%
1907     \expandafter\bbl@stringdef
1908     \csname\bbl@LC\expandafter\endcsname\expandafter{\BabelString}}}

```

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

```

1909 \ifx\bbl@opt@strings\relax
1910   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1911   \bbl@patchuclc
1912   \let\bbl@encoded\relax
1913   \def\bbl@encoded@uclc#1{%
1914     \@inmathwarn#1%

```

```

1915 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1916 \expandafter\ifx\csname ?\string#1\endcsname\relax
1917 \TextSymbolUnavailable#1%
1918 \else
1919 \csname ?\string#1\endcsname
1920 \fi
1921 \else
1922 \csname\cf@encoding\string#1\endcsname
1923 \fi}
1924 \else
1925 \def\bbl@scset#1#2{\def#1{#2}}
1926 \fi

```

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

```

1927 <<(*Macros local to BabelCommands)>> ≡
1928 \def\SetStringLoop##1##2{%
1929 \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1930 \count@\z@
1931 \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1932 \advance\count@\@ne
1933 \toks@\expandafter{\bbl@tempa}%
1934 \bbl@exp{%
1935 \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1936 \count@=\the\count@\relax}}}%
1937 <</Macros local to BabelCommands>>

```

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

```

1938 \def\bbl@aftercmds#1{%
1939 \toks@\expandafter{\bbl@scafter#1}%
1940 \xdef\bbl@scafter{\the\toks@}}

```

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

```

1941 <<(*Macros local to BabelCommands)>> ≡
1942 \newcommand\SetCase[3][]{%
1943 \bbl@patchuclc
1944 \bbl@forlang\bbl@tempa{%
1945 \bbl@carg\bbl@encstring{\bbl@tempa @bbl@uclc}{\bbl@tempa##1}%
1946 \bbl@carg\bbl@encstring{\bbl@tempa @bbl@uc}{##2}%
1947 \bbl@carg\bbl@encstring{\bbl@tempa @bbl@lc}{##3}}}%
1948 <</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.

```

1949 <<(*Macros local to BabelCommands)>> ≡
1950 \newcommand\SetHyphenMap[1]{%
1951 \bbl@forlang\bbl@tempa{%
1952 \expandafter\bbl@stringdef
1953 \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1954 <</Macros local to BabelCommands>>

```

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

```

1955 \newcommand\BabelLower[2]{% one to one.
1956 \ifnum\lccode#1=#2\else
1957 \babel@savevariable{\lccode#1}%
1958 \lccode#1=#2\relax
1959 \fi}
1960 \newcommand\BabelLowerMM[4]{% many-to-many

```



```

1961 \@tempcnta=#1\relax
1962 \@tempcntb=#4\relax
1963 \def\bbl@tempa{%
1964   \ifnum\@tempcnta>#2\else
1965     \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1966     \advance\@tempcnta#3\relax
1967     \advance\@tempcntb#3\relax
1968     \expandafter\bbl@tempa
1969   \fi}%
1970 \bbl@tempa}
1971 \newcommand\BabelLowerM0[4]{% many-to-one
1972   \@tempcnta=#1\relax
1973   \def\bbl@tempa{%
1974     \ifnum\@tempcnta>#2\else
1975       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1976       \advance\@tempcnta#3
1977       \expandafter\bbl@tempa
1978     \fi}%
1979   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1980 <<(*More package options)> \equiv
1981 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1982 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1983 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1984 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1985 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1986 <</More package options>

```

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

```

1987 \AtEndOfPackage{%
1988   \ifx\bbl@opt@hyphenmap\undefined
1989     \bbl@xin@{,}{\bbl@language@opts}%
1990     \chardef\bbl@opt@hyphenmap\ifin4\else\@ne\fi
1991   \fi}

```

This sections ends with a general tool for resetting the caption names with a unique interface. With the old way, which mixes the switcher and the string, we convert it to the new one, which separates these two steps.

```

1992 \newcommand\setlocalecaption{% TODO. Catch typos.
1993   \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
1994 \def\bbl@setcaption@x#1#2#3{% language caption-name string
1995   \bbl@trim@def\bbl@tempa{#2}%
1996   \bbl@xin@{.template}{\bbl@tempa}%
1997   \ifin@
1998     \bbl@ini@captions@template{#3}{#1}%
1999   \else
2000     \edef\bbl@tempd{%
2001       \expandafter\expandafter\expandafter
2002       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2003     \bbl@xin@
2004     {\expandafter\string\csname #2name\endcsname}%
2005     {\bbl@tempd}%
2006     \ifin@ % Renew caption
2007       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2008     \ifin@
2009       \bbl@exp{%
2010         \\bbl@ifsamestring{\bbl@tempa}{\language@name}%
2011         {\bbl@scset\<#2name>\<#1#2name>}%
2012         {}}%
2013       \else % Old way converts to new way
2014         \bbl@ifunset{#1#2name}%
2015         {\bbl@exp{%
2016           \\bbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%

```

```

2017         \bbl@ifsamestring{\bbl@tempa}{\language}%
2018         {\def\<#2name>{\<#1#2name>}}}%
2019         {}}}%
2020     {}%
2021     \fi
2022     \else
2023         \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2024         \ifin@ % New way
2025             \bbl@exp{%
2026                 \bbl@add\<captions#1>{\bbl@scset\<#2name>\<#1#2name>}}%
2027                 \bbl@ifsamestring{\bbl@tempa}{\language}%
2028                 {\bbl@scset\<#2name>\<#1#2name>}}%
2029                 {}}}%
2030     \else % Old way, but defined in the new way
2031         \bbl@exp{%
2032             \bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}}%
2033             \bbl@ifsamestring{\bbl@tempa}{\language}%
2034             {\def\<#2name>{\<#1#2name>}}}%
2035             {}}}%
2036     \fi%
2037     \fi
2038     \@namedef{#1#2name}{#3}%
2039     \toks@\expandafter{\bbl@captionslist}%
2040     \bbl@exp{\in@{\<#2name>}{\the\toks@}}%
2041     \ifin@\else
2042         \bbl@exp{\bbl@add\bbl@captionslist{\<#2name>}}%
2043         \bbl@toggle\bbl@captionslist
2044     \fi
2045     \fi}
2046 % \def\bbl@setcaption@s#1#2#3{} % TODO. Not yet implemented (w/o 'name')

```

## 4.11 Macros common to a number of languages

`\set@low@box` The following macro is used to lower quotes to the same level as the comma. It prepares its argument in box register 0.

```

2047 \bbl@trace{Macros related to glyphs}
2048 \def\set@low@box#1{\setbox\tw\hbox{,}\setbox\z\hbox{#1}%
2049     \dimen\z\ht\z@ \advance\dimen\z@ -\ht\tw@%
2050     \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.

```

2051 \def\save@sf@q#1{\leavevmode
2052     \begingroup
2053     \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2054     \endgroup}

```

## 4.12 Making glyphs available

This section makes a number of glyphs available that either do not exist in the OT1 encoding and have to be ‘faked’, or that are not accessible through T1enc.def.

### 4.12.1 Quotation marks

`\quotedblbase` In the T1 encoding the opening double quote at the baseline is available as a separate character, accessible via `\quotedblbase`. In the OT1 encoding it is not available, therefore we make it available by lowering the normal open quote character to the baseline.

```

2055 \ProvideTextCommand{\quotedblbase}{OT1}{%
2056     \save@sf@q{\set@low@box{\textquotedblright\}%
2057         \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.

```

2058 \ProvideTextCommandDefault{\quotedblbase}{%
2059     \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```
2060 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2061   \save@sf@q{\set@low@box{\textquoteright\}%
2062     \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.

```
2063 \ProvideTextCommandDefault{\quotesinglbase}{%
2064   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2065 \ProvideTextCommand{\guillemetleft}{OT1}{%
2066   \ifmmode
2067     \ll
2068   \else
2069     \save@sf@q{\nobreak
2070       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2071   \fi}
2072 \ProvideTextCommand{\guillemetright}{OT1}{%
2073   \ifmmode
2074     \gg
2075   \else
2076     \save@sf@q{\nobreak
2077       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2078   \fi}
2079 \ProvideTextCommand{\guillemotleft}{OT1}{%
2080   \ifmmode
2081     \ll
2082   \else
2083     \save@sf@q{\nobreak
2084       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2085   \fi}
2086 \ProvideTextCommand{\guillemotright}{OT1}{%
2087   \ifmmode
2088     \gg
2089   \else
2090     \save@sf@q{\nobreak
2091       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2092   \fi}
```

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

```
2093 \ProvideTextCommandDefault{\guillemetleft}{%
2094   \UseTextSymbol{OT1}{\guillemetleft}}
2095 \ProvideTextCommandDefault{\guillemetright}{%
2096   \UseTextSymbol{OT1}{\guillemetright}}
2097 \ProvideTextCommandDefault{\guillemotleft}{%
2098   \UseTextSymbol{OT1}{\guillemotleft}}
2099 \ProvideTextCommandDefault{\guillemotright}{%
2100   \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
2101 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2102   \ifmmode
2103     <%
2104   \else
2105     \save@sf@q{\nobreak
2106       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2107   \fi}
2108 \ProvideTextCommand{\guilsinglright}{OT1}{%
2109   \ifmmode
2110     >%
2111   \else
2112     \save@sf@q{\nobreak
```

```

2113      \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2114  \fi}

```

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

```

2115 \ProvideTextCommandDefault{\guilsinglleft}{%
2116  \UseTextSymbol{OT1}{\guilsinglleft}}
2117 \ProvideTextCommandDefault{\guilsinglright}{%
2118  \UseTextSymbol{OT1}{\guilsinglright}}

```

#### 4.12.2 Letters

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

```

2119 \DeclareTextCommand{\ij}{OT1}{%
2120  i\kern-0.02em\bbl@allowhyphens j}
2121 \DeclareTextCommand{\IJ}{OT1}{%
2122  I\kern-0.02em\bbl@allowhyphens J}
2123 \DeclareTextCommand{\ij}{T1}{\char188}
2124 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2125 \ProvideTextCommandDefault{\ij}{%
2126  \UseTextSymbol{OT1}{\ij}}
2127 \ProvideTextCommandDefault{\IJ}{%
2128  \UseTextSymbol{OT1}{\IJ}}

```

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

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

```

2129 \def\crrtic@{\hrule height0.1ex width0.3em}
2130 \def\crttic@{\hrule height0.1ex width0.33em}
2131 \def\ddj@{%
2132  \setbox0\hbox{d}\dimen@=\ht0
2133  \advance\dimen@1ex
2134  \dimen@.45\dimen@
2135  \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2136  \advance\dimen@ii.5ex
2137  \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2138 \def\DDJ@{%
2139  \setbox0\hbox{D}\dimen@=.55\ht0
2140  \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2141  \advance\dimen@ii.15ex % correction for the dash position
2142  \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2143  \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2144  \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2145 %
2146 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2147 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2148 \ProvideTextCommandDefault{\dj}{%
2149  \UseTextSymbol{OT1}{\dj}}
2150 \ProvideTextCommandDefault{\DJ}{%
2151  \UseTextSymbol{OT1}{\DJ}}

```

`\SS` For the T1 encoding `\SS` is defined and selects a specific glyph from the font, but for other encodings it is not available. Therefore we make it available here.

```

2152 \DeclareTextCommand{\SS}{OT1}{\SS}
2153 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}

```

### 4.12.3 Shorthands for quotation marks

Shorthands are provided for a number of different quotation marks, which make them usable both outside and inside mathmode. They are defined with `\ProvideTextCommandDefault`, but this is very likely not required because their definitions are based on encoding-dependent macros.

`\glq` The ‘german’ single quotes.

```
\grq 2154 \ProvideTextCommandDefault{\glq}{%
      2155 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2156 \ProvideTextCommand{\grq}{T1}{%
2157 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}%
2158 \ProvideTextCommand{\grq}{TU}{%
2159 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2160 \ProvideTextCommand{\grq}{OT1}{%
2161 \save@sf@q{\kern-.0125em
2162 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2163 \kern.07em\relax}}
2164 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

`\glqq` The ‘german’ double quotes.

```
\grqq 2165 \ProvideTextCommandDefault{\glqq}{%
      2166 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2167 \ProvideTextCommand{\grqq}{T1}{%
2168 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2169 \ProvideTextCommand{\grqq}{TU}{%
2170 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2171 \ProvideTextCommand{\grqq}{OT1}{%
2172 \save@sf@q{\kern-.07em
2173 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2174 \kern.07em\relax}}
2175 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq` The ‘french’ single guillemets.

```
\frq 2176 \ProvideTextCommandDefault{\flq}{%
      2177 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}%
2178 \ProvideTextCommandDefault{\frq}{%
2179 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq 2180 \ProvideTextCommandDefault{\flqq}{%
      2181 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}%
2182 \ProvideTextCommandDefault{\frqq}{%
2183 \textormath{\guillemetright}{\mbox{\guillemetright}}}
```

### 4.12.4 Umlauts and tremas

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

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

```
\umlautlow 2184 \def\umlauthigh{%
2185 \def\bbl@umlauta##1{\leavevmode\bgroup%
2186 \accent\csname\fontencoding dqpos\endcsname
2187 ##1\bbl@allowhyphens\egroup}%
2188 \let\bbl@umlaute\bbl@umlauta}
2189 \def\umlautlow{%
```

```

2190 \def\bbl@umlauta{\protect\lower@umlaut}}
2191 \def\umlautelow{%
2192 \def\bbl@umlaute{\protect\lower@umlaut}}
2193 \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.

```

2194 \expandafter\ifx\csname U@D\endcsname\relax
2195 \csname newdimen\endcsname\U@D
2196 \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.

```

2197 \def\lower@umlaut#1{%
2198 \leavevmode\bggroup
2199 \U@D 1ex%
2200 {\setbox\z@\hbox{%
2201 \char\csname\fontencoding dqpos\endcsname}%
2202 \dimen@ -.45ex\advance\dimen@\ht\z@
2203 \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2204 \accent\csname\fontencoding dqpos\endcsname
2205 \fontdimen5\font\U@D #1%
2206 \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).

```

2207 \AtBeginDocument{%
2208 \DeclareTextCompositeCommand{"}{OT1}{a}{\bbl@umlauta{a}}%
2209 \DeclareTextCompositeCommand{"}{OT1}{e}{\bbl@umlaute{e}}%
2210 \DeclareTextCompositeCommand{"}{OT1}{i}{\bbl@umlaute{i}}%
2211 \DeclareTextCompositeCommand{"}{OT1}{\i}{\bbl@umlaute{i}}%
2212 \DeclareTextCompositeCommand{"}{OT1}{o}{\bbl@umlauta{o}}%
2213 \DeclareTextCompositeCommand{"}{OT1}{u}{\bbl@umlauta{u}}%
2214 \DeclareTextCompositeCommand{"}{OT1}{A}{\bbl@umlauta{A}}%
2215 \DeclareTextCompositeCommand{"}{OT1}{E}{\bbl@umlaute{E}}%
2216 \DeclareTextCompositeCommand{"}{OT1}{I}{\bbl@umlaute{I}}%
2217 \DeclareTextCompositeCommand{"}{OT1}{O}{\bbl@umlauta{O}}%
2218 \DeclareTextCompositeCommand{"}{OT1}{U}{\bbl@umlauta{U}}%

```

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

```

2219 \ifx\l@english\@undefined
2220 \chardef\l@english\z@
2221 \fi
2222 % The following is used to cancel rules in ini files (see Amharic).
2223 \ifx\l@unhyphenated\@undefined
2224 \newlanguage\l@unhyphenated
2225 \fi

```

## 4.13 Layout

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

```

2226 \bbl@trace{Bidi layout}
2227 \providecommand\IfBabelLayout[3]{#3}%
2228 <-core>
2229 \newcommand\BabelPatchSection[1]{%
2230   \@ifundefined{#1}{}{%
2231     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2232     \@namedef{#1}{%
2233       \@ifstar{\bbl@presec@s{#1}}{%
2234         {\@dblarg{\bbl@presec@x{#1}}}}}%
2235 \def\bbl@presec@x#1[#2]#3{%
2236   \bbl@exp{%
2237     \\select@language@x{\bbl@main@language}%
2238     \\bbl@cs{sspre@#1}%
2239     \\bbl@cs{ss@#1}%
2240     [\\foreignlanguage{\language}{\unexpanded{#2}}}%
2241     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2242     \\select@language@x{\language}}}%
2243 \def\bbl@presec@s#1#2{%
2244   \bbl@exp{%
2245     \\select@language@x{\bbl@main@language}%
2246     \\bbl@cs{sspre@#1}%
2247     \\bbl@cs{ss@#1}*%
2248     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
2249     \\select@language@x{\language}}}%
2250 \IfBabelLayout{sectioning}%
2251   {\BabelPatchSection{part}%
2252    \BabelPatchSection{chapter}%
2253    \BabelPatchSection{section}%
2254    \BabelPatchSection{subsection}%
2255    \BabelPatchSection{subsubsection}%
2256    \BabelPatchSection{paragraph}%
2257    \BabelPatchSection{subparagraph}%
2258    \def\babel@toc#1{%
2259      \select@language@x{\bbl@main@language}}}%
2260 \IfBabelLayout{captions}%
2261   {\BabelPatchSection{caption}}}%
2262 <+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.

```

2263 \bbl@trace{Input engine specific macros}
2264 \ifcase\bbl@engine
2265   \input txtbabel.def
2266 \or
2267   \input luababel.def
2268 \or
2269   \input xebabel.def
2270 \fi
2271 \providecommand\babelfont{%
2272   \bbl@error
2273   {This macro is available only in LuaLaTeX and XeLaTeX.}%
2274   {Consider switching to these engines.}}
2275 \providecommand\babelprehyphenation{%
2276   \bbl@error
2277   {This macro is available only in LuaLaTeX.}%
2278   {Consider switching to that engine.}}
2279 \ifx\babelposthyphenation\@undefined
2280   \let\babelposthyphenation\babelprehyphenation
2281   \let\babelpatterns\babelprehyphenation
2282   \let\babelcharproperty\babelprehyphenation
2283 \fi

```

## 4.15 Creating and modifying languages

Continue with  $\LaTeX$  only.

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

```
2284 </package | core>
2285 <{*package}>
2286 \bbl@trace{Creating languages and reading ini files}
2287 \let\bbl@extend@ini\@gobble
2288 \newcommand\babelprovide[2][]{%
2289   \let\bbl@savelangname\language
2290   \edef\bbl@savelocaleid{\the\localeid}%
2291   % Set name and locale id
2292   \edef\language{#2}%
2293   \bbl@id@assign
2294   % Initialize keys
2295   \bbl@vforeach{captions,date,import,main,script,language,%
2296     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2297     mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2298     Alph,labels,labels*,calendar,date,casing}%
2299     {\bbl@csarg\let{KVP@##1}\@nnil}%
2300   \global\let\bbl@release@transforms\@empty
2301   \let\bbl@calendars\@empty
2302   \global\let\bbl@inidata\@empty
2303   \global\let\bbl@extend@ini\@gobble
2304   \global\let\bbl@included@inis\@empty
2305   \gdef\bbl@key@list{;}%
2306   \bbl@forkv{#1}{%
2307     \in@{/}{##1}% With /, (re)sets a value in the ini
2308     \ifin@
2309       \global\let\bbl@extend@ini\bbl@extend@ini@aux
2310       \bbl@renewinikey##1\@{##2}%
2311     \else
2312       \bbl@csarg\ifx{KVP@##1}\@nnil\else
2313         \bbl@error
2314           {Unknown key '##1' in \string\babelprovide}%
2315           {See the manual for valid keys}%
2316       \fi
2317       \bbl@csarg\def{KVP@##1}{##2}%
2318     \fi}%
2319   \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2320   \bbl@ifunset{date#2}\z@{\bbl@ifunset{\bbl@llevel@#2}\@ne\@tw@}%
2321   % == init ==
2322   \ifx\bbl@screset\undefined
2323     \bbl@ldfinit
2324   \fi
2325   % == date (as option) ==
2326   % \ifx\bbl@KVP@date\@nnil\else
2327   % \fi
2328   % ==
2329   \let\bbl@lbkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2330   \ifcase\bbl@howloaded
2331     \let\bbl@lbkflag\@empty % new
2332   \else
2333     \ifx\bbl@KVP@hyphenrules\@nnil\else
2334       \let\bbl@lbkflag\@empty
2335     \fi
2336     \ifx\bbl@KVP@import\@nnil\else
2337       \let\bbl@lbkflag\@empty
2338     \fi
2339   \fi
2340   % == import, captions ==
```



```

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

```

2400 \bbl@load@basic{#2}%
2401 % == script, language ==
2402 % Override the values from ini or defines them
2403 \ifx\bbl@KVP@script\@nnil\else
2404   \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2405 \fi
2406 \ifx\bbl@KVP@language\@nnil\else
2407   \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2408 \fi
2409 \ifcase\bbl@engine\or
2410   \bbl@ifunset{\bbl@chrng@\languagename}{}%
2411   {\directlua{
2412     Babel.set_chranges_b('\bbl@cl{sbc}p', '\bbl@cl{chrng}') }}%
2413 \fi
2414 % == onchar ==
2415 \ifx\bbl@KVP@onchar\@nnil\else
2416   \bbl@luahyphenate
2417   \bbl@exp{%
2418     \AddToHook{env/document/before}{\select@language{#2}}}%
2419   \directlua{
2420     if Babel.locale_mapped == nil then
2421       Babel.locale_mapped = true
2422       Babel.linebreaking.add_before(Babel.locale_map, 1)
2423       Babel.loc_to_scr = {}
2424       Babel.chr_to_loc = Babel.chr_to_loc or {}
2425     end
2426     Babel.locale_props[\the\localeid].letters = false
2427   }%
2428   \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
2429   \ifin@
2430     \directlua{
2431       Babel.locale_props[\the\localeid].letters = true
2432     }%
2433   \fi
2434   \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2435   \ifin@
2436     \ifx\bbl@starthyphens\undefined % Needed if no explicit selection
2437       \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
2438     \fi
2439     \bbl@exp{\bbl@add\bbl@starthyphens
2440       {\bbl@patterns@lua{\languagename}}}%
2441     % TODO - error/warning if no script
2442     \directlua{
2443       if Babel.script_blocks['\bbl@cl{sbc}p'] then
2444         Babel.loc_to_scr[\the\localeid] =
2445           Babel.script_blocks['\bbl@cl{sbc}p']
2446         Babel.locale_props[\the\localeid].lc = \the\localeid\space
2447         Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space
2448       end
2449     }%
2450   \fi
2451   \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2452   \ifin@
2453     \bbl@ifunset{\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}}%
2454     \bbl@ifunset{\bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}}%
2455     \directlua{
2456       if Babel.script_blocks['\bbl@cl{sbc}p'] then
2457         Babel.loc_to_scr[\the\localeid] =
2458           Babel.script_blocks['\bbl@cl{sbc}p']
2459       end}%
2460   \ifx\bbl@mapselect\undefined % TODO. almost the same as mapfont
2461     \AtBeginDocument{%
2462       \bbl@patchfont{\bbl@mapselect}}%

```

```

2463     {\selectfont}}%
2464 \def\bb@mapselect{%
2465     \let\bb@mapselect\relax
2466     \edef\bb@prefontid{\fontid\font}}%
2467 \def\bb@mapdir##1{%
2468     {\def\language{##1}%
2469     \let\bb@ifrestoring\@firstoftwo % To avoid font warning
2470     \bb@switchfont
2471     \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2472     \directlua{
2473         Babel.locale_props[\the\csname bb@id@##1\endcsname]%
2474         [\bb@prefontid] = \fontid\font\space}%
2475     \fi}}%
2476 \fi
2477 \bb@exp{\bb@add\bb@mapselect{\bb@mapdir{\language}}}%
2478 \fi
2479 % TODO - catch non-valid values
2480 \fi
2481 % == mapfont ==
2482 % For bidi texts, to switch the font based on direction
2483 \ifx\bb@KVP@mapfont\@nnil\else
2484     \bb@ifsamestring{\bb@KVP@mapfont}{direction}}%
2485     {\bb@error{Option '\bb@KVP@mapfont' unknown for\%
2486         mapfont. Use 'direction'.%
2487         {See the manual for details.}}}%
2488     \bb@ifunset{\bb@lsys\language}{\bb@provide@lsys\language}}%
2489     \bb@ifunset{\bb@wdir\language}{\bb@provide@dirs\language}}%
2490 \ifx\bb@mapselect\undefined % TODO. See onchar.
2491     \AtBeginDocument{%
2492         \bb@patchfont{\bb@mapselect}}%
2493     {\selectfont}}%
2494 \def\bb@mapselect{%
2495     \let\bb@mapselect\relax
2496     \edef\bb@prefontid{\fontid\font}}%
2497 \def\bb@mapdir##1{%
2498     {\def\language{##1}%
2499     \let\bb@ifrestoring\@firstoftwo % avoid font warning
2500     \bb@switchfont
2501     \directlua{Babel.fontmap
2502         [\the\csname bb@wdir@##1\endcsname]%
2503         [\bb@prefontid]=\fontid\font}}}%
2504 \fi
2505 \bb@exp{\bb@add\bb@mapselect{\bb@mapdir{\language}}}%
2506 \fi
2507 % == Line breaking: intraspace, intrapenalty ==
2508 % For CJK, East Asian, Southeast Asian, if interspace in ini
2509 \ifx\bb@KVP@intraspace\@nnil\else % We can override the ini or set
2510     \bb@csarg\edef{intsp@#2}{\bb@KVP@intraspace}%
2511 \fi
2512 \bb@provide@intraspace
2513 % == Line breaking: CJK quotes == TODO -> @extras
2514 \ifcase\bb@engine\or
2515     \bb@xin@{/c}{/\bb@cl{lnbrk}}%
2516 \ifin@
2517     \bb@ifunset{\bb@quote\language}}%
2518     {\directlua{
2519         Babel.locale_props[\the\localeid].cjk_quotes = {}
2520         local cs = 'op'
2521         for c in string.utfvalues(
2522             [[\csname bb@quote\language\endcsname]]) do
2523             if Babel.cjk_characters[c].c == 'qu' then
2524                 Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2525             end

```

```

2526         cs = ( cs == 'op') and 'cl' or 'op'
2527     end
2528 }}%
2529 \fi
2530 \fi
2531 % == Line breaking: justification ==
2532 \ifx\bb1@KVP@justification\@nnil\else
2533     \let\bb1@KVP@linebreaking\bb1@KVP@justification
2534 \fi
2535 \ifx\bb1@KVP@linebreaking\@nnil\else
2536     \bb1@xin@{,\bb1@KVP@linebreaking,}%
2537     {,elongated,kashida,cjk,padding,unhyphenated,}%
2538 \ifin@
2539     \bb1@csarg\xdef
2540     {\lnbrk@\language\name}{\expandafter\@car\bb1@KVP@linebreaking\@nil}%
2541 \fi
2542 \fi
2543 \bb1@xin@{/e}{/\bb1@cl{\lnbrk}}%
2544 \ifin@\else\bb1@xin@{/k}{/\bb1@cl{\lnbrk}}\fi
2545 \ifin@\bb1@arabicjust\fi
2546 \bb1@xin@{/p}{/\bb1@cl{\lnbrk}}%
2547 \ifin@\AtBeginDocument{\@nameuse{\bb1@tibetanjust}}\fi
2548 % == Line breaking: hyphenate.other.(locale|script) ==
2549 \ifx\bb1@lbfkflag\@empty
2550     \bb1@ifunset{\bb1@hyotl@\language\name}{}%
2551     {\bb1@csarg\bb1@replace{\hyotl@\language\name}{ }{ },}%
2552     \bb1@startcommands*{\language\name}{}%
2553     \bb1@csarg\bb1@foreach{\hyotl@\language\name}{%
2554         \ifcase\bb1@engine
2555             \ifnum##1<257
2556                 \SetHyphenMap{\BabelLower{##1}{##1}}%
2557             \fi
2558             \else
2559                 \SetHyphenMap{\BabelLower{##1}{##1}}%
2560             \fi}%
2561     \bb1@endcommands}%
2562 \bb1@ifunset{\bb1@hyots@\language\name}{}%
2563     {\bb1@csarg\bb1@replace{\hyots@\language\name}{ }{ },}%
2564     \bb1@csarg\bb1@foreach{\hyots@\language\name}{%
2565         \ifcase\bb1@engine
2566             \ifnum##1<257
2567                 \global\lccode##1=##1\relax
2568             \fi
2569             \else
2570                 \global\lccode##1=##1\relax
2571             \fi}%
2572 \fi
2573 % == Counters: maparabic ==
2574 % Native digits, if provided in ini (TeX level, xe and lua)
2575 \ifcase\bb1@engine\else
2576     \bb1@ifunset{\bb1@dgnat@\language\name}{}%
2577     {\expandafter\ifx\csname \bb1@dgnat@\language\name\endcsname\@empty\else
2578         \expandafter\expandafter\expandafter
2579         \bb1@setdigits\csname \bb1@dgnat@\language\name\endcsname
2580         \ifx\bb1@KVP@maparabic\@nnil\else
2581             \ifx\bb1@latinarabic\@undefined
2582                 \expandafter\let\expandafter\@arabic
2583                 \csname \bb1@counter@\language\name\endcsname
2584             \else % ie, if layout=counters, which redefines \@arabic
2585                 \expandafter\let\expandafter\bb1@latinarabic
2586                 \csname \bb1@counter@\language\name\endcsname
2587             \fi
2588         \fi

```

```

2589     \fi}%
2590 \fi
2591 % == Counters: mapdigits ==
2592 % > luababel.def
2593 % == Counters: alph, Alph ==
2594 \ifx\bbl@KVP@alph@nnil\else
2595     \bbl@exp{%
2596         \\bbl@add\<bbl@preextras@\language\>{%
2597             \\babel@save\\@alph
2598             \let\\@alph\<bbl@cntr@\bbl@KVP@alph @\language\>}}%
2599 \fi
2600 \ifx\bbl@KVP@Alph@nnil\else
2601     \bbl@exp{%
2602         \\bbl@add\<bbl@preextras@\language\>{%
2603             \\babel@save\\@Alph
2604             \let\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\language\>}}%
2605 \fi
2606 % == Casing ==
2607 \bbl@exp{\def\<bbl@casing@\language\>%
2608     {\<bbl@lbcpr@\language\>%
2609     \ifx\bbl@KVP@casing@nnil\else-x-\bbl@KVP@casing\fi}}%
2610 % == Calendars ==
2611 \ifx\bbl@KVP@calendar@nnil
2612     \edef\bbl@KVP@calendar{\bbl@cl{calpr}}%
2613 \fi
2614 \def\bbl@tempe##1 ##2\@{% Get first calendar
2615     \def\bbl@tempa{##1}}%
2616     \bbl@exp{\\bbl@tempe\bbl@KVP@calendar\space\\@}%
2617 \def\bbl@tempe##1.##2.##3\@{%
2618     \def\bbl@tempc{##1}%
2619     \def\bbl@tempb{##2}}%
2620 \expandafter\bbl@tempe\bbl@tempa..@@
2621 \bbl@csarg\edef{calpr@\language\>}{%
2622     \ifx\bbl@tempc@empty\else
2623         calendar=\bbl@tempc
2624     \fi
2625     \ifx\bbl@tempb@empty\else
2626         ,variant=\bbl@tempb
2627     \fi}%
2628 % == engine specific extensions ==
2629 % Defined in XXXbabel.def
2630 \bbl@provide@extra{#2}%
2631 % == require.babel in ini ==
2632 % To load or reload the babel-*.tex, if require.babel in ini
2633 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
2634     \bbl@ifunset{\bbl@rqtex@\language\>}{}%
2635     {\expandafter\ifx\csname\bbl@rqtex@\language\>\endcsname\@empty\else
2636         \let\BabelBeforeIni@gobbletwo
2637         \chardef\atcatcode=\catcode\@
2638         \catcode\@=11\relax
2639         \bbl@input@texini{\bbl@cs{rqtex@\language\>}}%
2640         \catcode\@=\atcatcode
2641         \let\atcatcode\relax
2642         \global\bbl@csarg\let{rqtex@\language\>}\relax
2643     \fi}%
2644 \bbl@foreach\bbl@calendars{%
2645     \bbl@ifunset{\bbl@ca##1}{%
2646         \chardef\atcatcode=\catcode\@
2647         \catcode\@=11\relax
2648         \InputIfFileExists{babel-ca-##1.tex}{}}%
2649     \catcode\@=\atcatcode
2650     \let\atcatcode\relax}%
2651 {}%

```

```

2652 \fi
2653 % == frenchspacing ==
2654 \ifcase\bbbl@howloaded\in@true\else\in@false\fi
2655 \ifin@ \else\bbbl@xin@{typography/frenchspacing}{\bbbl@key@list}\fi
2656 \ifin@
2657   \bbbl@extras@wrap{\bbbl@pre@fs}%
2658   {\bbbl@pre@fs}%
2659   {\bbbl@post@fs}%
2660 \fi
2661 % == transforms ==
2662 % > luababel.def
2663 % == main ==
2664 \ifx\bbbl@KVP@main@\nnil % Restore only if not 'main'
2665   \let\languagename\bbbl@savelangname
2666   \chardef\localeid\bbbl@savelocaleid\relax
2667 \fi
2668 % == hyphenrules (apply if current) ==
2669 \ifx\bbbl@KVP@hyphenrules@\nnil\else
2670   \ifnum\bbbl@savelocaleid=\localeid
2671     \language\@nameuse{1\languagename}%
2672   \fi
2673 \fi}

```

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

```

2674 \def\bbbl@provide@new#1{%
2675   \@namedef{date#1}{}}% marks lang exists - required by \StartBabelCommands
2676   \@namedef{extras#1}{}}%
2677   \@namedef{noextras#1}{}}%
2678   \bbbl@startcommands*{#1}{captions}%
2679   \ifx\bbbl@KVP@captions@\nnil % and also if import, implicit
2680     \def\bbbl@tempb##1{% elt for \bbbl@captionslist
2681       \ifx##1\@empty\else
2682         \bbbl@exp{%
2683           \SetString\##1{%
2684             \bbbl@nocaption{\bbbl@stripslash##1}{#1\bbbl@stripslash##1}}}%
2685         \expandafter\bbbl@tempb
2686       \fi}%
2687   \expandafter\bbbl@tempb\bbbl@captionslist\@empty
2688   \else
2689     \ifx\bbbl@initoload\relax
2690       \bbbl@read@ini{\bbbl@KVP@captions}2% % Here letters cat = 11
2691     \else
2692       \bbbl@read@ini{\bbbl@initoload}2% % Same
2693     \fi
2694   \fi
2695   \StartBabelCommands*{#1}{date}%
2696   \ifx\bbbl@KVP@date@\nnil
2697     \bbbl@exp{%
2698       \SetString\today{\bbbl@nocaption{today}{#1today}}}%
2699   \else
2700     \bbbl@savetoday
2701     \bbbl@savedate
2702   \fi
2703   \bbbl@endcommands
2704   \bbbl@load@basic{#1}%
2705   % == hyphenmins == (only if new)
2706   \bbbl@exp{%
2707     \gdef\<#1hyphenmins>{%
2708       {\bbbl@ifunset{\bbbl@lfthm@#1}{2}{\bbbl@cs{lfthm@#1}}}%
2709       {\bbbl@ifunset{\bbbl@rgthm@#1}{3}{\bbbl@cs{rgthm@#1}}}}}%
2710   % == hyphenrules (also in renew) ==
2711   \bbbl@provide@hyphens{#1}%

```

```

2712 \ifx\bb1@KVP@main\@nnil\else
2713 \expandafter\main@language\expandafter{#1}%
2714 \fi}
2715 %
2716 \def\bb1@provide@renew#1{%
2717 \ifx\bb1@KVP@captions\@nnil\else
2718 \StartBabelCommands*{#1}{captions}%
2719 \bb1@read@ini{\bb1@KVP@captions}2% % Here all letters cat = 11
2720 \EndBabelCommands
2721 \fi
2722 \ifx\bb1@KVP@date\@nnil\else
2723 \StartBabelCommands*{#1}{date}%
2724 \bb1@savetoday
2725 \bb1@savedate
2726 \EndBabelCommands
2727 \fi
2728 % == hyphenrules (also in new) ==
2729 \ifx\bb1@lbfkflag\@empty
2730 \bb1@provide@hyphens{#1}%
2731 \fi}

```

Load the basic parameters (ids, typography, counters, and a few more), while captions and dates are left out. But it may happen some data has been loaded before automatically, so we first discard the saved values. (TODO. But preserving previous values would be useful.)

```

2732 \def\bb1@load@basic#1{%
2733 \ifcase\bb1@howloaded\or\or
2734 \ifcase\csname \bb1@llevel@\language\endcsname
2735 \bb1@csarg\let\lname@\language\relax
2736 \fi
2737 \fi
2738 \bb1@ifunset{\bb1@lname@#1}%
2739 {\def\BabelBeforeIni##1##2{%
2740 \begingroup
2741 \let\bb1@ini@captions@aux\@gobbletwo
2742 \def\bb1@inidate ####1.####2.####3.####4\relax ####5####6}%
2743 \bb1@read@ini{##1}1%
2744 \ifx\bb1@initoload\relax\endinput\fi
2745 \endgroup}%
2746 \begingroup % boxed, to avoid extra spaces:
2747 \ifx\bb1@initoload\relax
2748 \bb1@input@texini{##1}%
2749 \else
2750 \setbox\z@\hbox{\BabelBeforeIni{\bb1@initoload}}}%
2751 \fi
2752 \endgroup}%
2753 {}}

```

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.

```

2754 \def\bb1@provide@hyphens#1{%
2755 \@tempcnta\m@ne % a flag
2756 \ifx\bb1@KVP@hyphenrules\@nnil\else
2757 \bb1@replace\bb1@KVP@hyphenrules{ }{,}%
2758 \bb1@foreach\bb1@KVP@hyphenrules{%
2759 \ifnum\@tempcnta=\m@ne % if not yet found
2760 \bb1@ifsamestring{##1}{+}%
2761 {\bb1@carg\addlanguage{l@##1}}%
2762 {}}%
2763 \bb1@ifunset{l@##1}% After a possible +
2764 {}%
2765 {\@tempcnta\@nameuse{l@##1}}%
2766 \fi}%
2767 \ifnum\@tempcnta=\m@ne
2768 \bb1@warning{%

```

```

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

```

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

```

2793 \def\bbl@input@texini#1{%
2794     \bbl@bsphack
2795     \bbl@exp{%
2796         \catcode`\%%=14 \catcode`\===0
2797         \catcode`\={1} \catcode`\}=2
2798         \lowercase{\InputIfFileExists{babel-#1.tex}{}}}%
2799         \catcode`\%%=\the\catcode`\%\relax
2800         \catcode`\===\the\catcode`\=\relax
2801         \catcode`\={\the\catcode`\}\relax
2802         \catcode`\}= \the\catcode`\}\relax}%
2803     \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.

```

2804 \def\bbl@iniline#1\bbl@iniline{%
2805     \ifnextchar[\bbl@inisect{\ifnextchar\bbl@iniskip\bbl@inistore}#1\@@}% ]
2806 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2807 \def\bbl@iniskip#1\@@{%           if starts with ;
2808 \def\bbl@inistore#1=#2\@@{%      full (default)
2809     \bbl@trim@def\bbl@tempa{#1}%
2810     \bbl@trim\toks@{#2}%
2811     \bbl@xin@{\bbl@section/\bbl@tempa}{\bbl@key@list}%
2812 \ifin\else
2813     \bbl@xin@{,identification/include.}%
2814     {\bbl@section/\bbl@tempa}%
2815     \ifin@\xdef\bbl@included@inis{\the\toks@}\fi
2816     \bbl@exp{%
2817         \\g@addto@macro\\bbl@inidata{%
2818             \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2819     \fi}
2820 \def\bbl@inistore@min#1=#2\@@{%   minimal (maybe set in \bbl@read@ini)
2821     \bbl@trim@def\bbl@tempa{#1}%
2822     \bbl@trim\toks@{#2}%
2823     \bbl@xin@{.identification.}{\bbl@section.}%
2824 \ifin@
2825     \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2826         \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%

```



```
2827 \fi}
```

Now, the ‘main loop’, which **\*\*must be executed inside a group\*\***. At this point, \bbl@inidata may contain data declared in \babelprovide, with ‘slashed’ keys. There are 3 steps: first read the ini file and store it; then traverse the stored values, and process some groups if required (date, captions, labels, counters); finally, ‘export’ some values by defining global macros (identification, typography, characters, numbers). The second argument is 0 when called to read the minimal data for fonts; with \babelprovide it’s either 1 or 2.

```
2828 \def\bbl@loop@ini{%
2829   \loop
2830     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2831     \endlinechar\m@ne
2832     \read\bbl@readstream to \bbl@line
2833     \endlinechar\^^M
2834     \ifx\bbl@line\empty\else
2835       \expandafter\bbl@inline\bbl@line\bbl@inline
2836     \fi
2837   \repeat}
2838 \ifx\bbl@readstream\undefined
2839   \csname newread\endcsname\bbl@readstream
2840 \fi
2841 \def\bbl@read@ini#1#2{%
2842   \global\let\bbl@extend@ini\@gobble
2843   \openin\bbl@readstream=babel-#1.ini
2844   \ifeof\bbl@readstream
2845     \bbl@error
2846     {There is no ini file for the requested language\\%
2847      (#1: \language). Perhaps you misspelled it or your\\%
2848      installation is not complete.}%
2849     {Fix the name or reinstall babel.}%
2850   \else
2851     % == Store ini data in \bbl@inidata ==
2852     \catcode\ [=12 \catcode\ ]=12 \catcode\ |=12 \catcode\ &=12
2853     \catcode\ ;=12 \catcode\ |=12 \catcode\ %=14 \catcode\ -=12
2854     \bbl@info{Importing
2855               \ifcase#2font and identification \or basic \fi
2856               data for \language\\%
2857               from babel-#1.ini. Reported}%
2858     \ifnum#2=\z@
2859       \global\let\bbl@inidata\empty
2860       \let\bbl@inistore\bbl@inistore@min % Remember it's local
2861     \fi
2862     \def\bbl@section{identification}%
2863     \bbl@exp{\bbl@inistore tag.ini=#1\\@@}%
2864     \bbl@inistore load.level=#2\\@@
2865     \bbl@loop@ini
2866     % == Process stored data ==
2867     \bbl@csarg\edef{lini@\language}{#1}%
2868     \bbl@read@ini@aux
2869     % == 'Export' data ==
2870     \bbl@ini@exports{#2}%
2871     \global\bbl@csarg\let{inidata@\language}\bbl@inidata
2872     \global\let\bbl@inidata\empty
2873     \bbl@exp{\bbl@add@list\bbl@ini@loaded{\language}}%
2874     \bbl@tglobal\bbl@ini@loaded
2875   \fi
2876   \closein\bbl@readstream}
2877 \def\bbl@read@ini@aux{%
2878   \let\bbl@savestrings\empty
2879   \let\bbl@savetoday\empty
2880   \let\bbl@savestate\empty
2881   \def\bbl@elt##1##2##3{%
2882     \def\bbl@section{##1}%
```

```

2883 \in@{=date.}{=##1}% Find a better place
2884 \ifin@
2885 \bbl@ifunset{bbl@inikv@##1}%
2886 {\bbl@ini@calendar{##1}}%
2887 {}%
2888 \fi
2889 \bbl@ifunset{bbl@inikv@##1}{}%
2890 {\csname bbl@inikv@##1\endcsname{##2}{##3}}%
2891 \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.

```

2892 \def\bbl@extend@ini@aux#1{%
2893 \bbl@startcommands*{#1}{captions}%
2894 % Activate captions/... and modify exports
2895 \bbl@csarg\def{inikv@captions.licr}##1##2{%
2896 \setlocalecaption{#1}{##1}{##2}}%
2897 \def\bbl@inikv@captions##1##2{%
2898 \bbl@ini@captions@aux{##1}{##2}}%
2899 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2900 \def\bbl@exportkey##1##2##3{%
2901 \bbl@ifunset{bbl@kv@##2}{}%
2902 {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2903 \bbl@exp{\global\let<bbl@##1@language>\<bbl@kv@##2>}}%
2904 \fi}}%
2905 % As with \bbl@read@ini, but with some changes
2906 \bbl@read@ini@aux
2907 \bbl@ini@exports\tw@
2908 % Update inidata@lang by pretending the ini is read.
2909 \def\bbl@elt##1##2##3{%
2910 \def\bbl@section{##1}%
2911 \bbl@inline##2=##3\bbl@inline}%
2912 \csname bbl@inidata@#1\endcsname
2913 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2914 \StartBabelCommands*{#1}{date}% And from the import stuff
2915 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2916 \bbl@savetoday
2917 \bbl@savestate
2918 \bbl@endcommands}

```

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

```

2919 \def\bbl@ini@calendar#1{%
2920 \lowercase{\def\bbl@tempa{=#1=}}%
2921 \bbl@replace\bbl@tempa{=date.gregorian}{}%
2922 \bbl@replace\bbl@tempa{=date.}{}%
2923 \in@{.licr}={#1=}%
2924 \ifin@
2925 \ifcase\bbl@engine
2926 \bbl@replace\bbl@tempa{.licr=}{}%
2927 \else
2928 \let\bbl@tempa\relax
2929 \fi
2930 \fi
2931 \ifx\bbl@tempa\relax\else
2932 \bbl@replace\bbl@tempa{=}{}%
2933 \ifx\bbl@tempa\@empty\else
2934 \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2935 \fi
2936 \bbl@exp{%
2937 \def<bbl@inikv@#1>#####1####2{%
2938 \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2939 \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).

```

2940 \def\bbl@renewinikey#1/#2\@#3{%
2941   \edef\bbl@tempa{\zap@space #1 \@empty}%   section
2942   \edef\bbl@tempb{\zap@space #2 \@empty}%   key
2943   \bbl@trim\toks@{#3}%                       value
2944   \bbl@exp{%
2945     \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2946     \\g@addto@macro\\bbl@inidata{%
2947       \\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.

```

2948 \def\bbl@exportkey#1#2#3{%
2949   \bbl@ifunset{bbl@kv@#2}%
2950     { \bbl@csarg\gdef{#1@\language@}{#3}}%
2951     { \expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
2952       \bbl@csarg\gdef{#1@\language@}{#3}%
2953       \else
2954         \bbl@exp{\global\let<bbl@#1@\language@>\<bbl@kv@#2>}%
2955         \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@inise), 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.

```

2956 \def\bbl@iniwarning#1{%
2957   \bbl@ifunset{bbl@kv@identification.warning#1}{}%
2958   { \bbl@warning{%
2959     From babel-\bbl@cs{lini@\language@}.ini:\%
2960     \bbl@cs{kv@identification.warning#1}\%
2961     Reported }}%
2962 %
2963 \let\bbl@release@transforms\@empty
2964 \def\bbl@ini@exports#1{%
2965   % Identification always exported
2966   \bbl@iniwarning}%
2967   \ifcase\bbl@engine
2968     \bbl@iniwarning{.pdflatex}%
2969   \or
2970     \bbl@iniwarning{.lualatex}%
2971   \or
2972     \bbl@iniwarning{.xelatex}%
2973   \fi%
2974   \bbl@exportkey{llevel}{identification.load.level}{}%
2975   \bbl@exportkey{elname}{identification.name.english}{}%
2976   \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%
2977     {\csname bbl@elname@\language@endcsname}}%
2978   \bbl@exportkey{tbc}{identification.tag.bcp47}{}%
2979   \bbl@exportkey{lbc}{identification.language.tag.bcp47}{}%
2980   % Somewhat hackish. TODO
2981   \bbl@exportkey{casing}{identification.language.tag.bcp47}{}%
2982   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2983   \bbl@exportkey{esname}{identification.script.name}{}%
2984   \bbl@exp{\\bbl@exportkey{sname}{identification.script.name.opentype}%
2985     {\csname bbl@esname@\language@endcsname}}%
2986   \bbl@exportkey{sbc}{identification.script.tag.bcp47}{}%
2987   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2988   \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2989   \bbl@exportkey{vbc}{identification.variant.tag.bcp47}{}%
2990   \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%

```

```

2991 \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
2992 \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
2993 % Also maps bcp47 -> languagename
2994 \ifbbl@bcptoname
2995 \bbl@csarg\xdef{bcp@map@{bbl@cl{tbc}}}{\languagename}%
2996 \fi
2997 % Conditional
2998 \ifnum#1>\z@ % 0 = only info, 1, 2 = basic, (re)new
2999 \bbl@exportkey{calpr}{date.calendar.preferred}{}%
3000 \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3001 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3002 \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3003 \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3004 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3005 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3006 \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3007 \bbl@exportkey{intsp}{typography.intraspaces}{}%
3008 \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3009 \bbl@exportkey{chrng}{characters.ranges}{}%
3010 \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
3011 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3012 \ifnum#1=\tw@ % only (re)new
3013 \bbl@exportkey{rqtex}{identification.require.babel}{}%
3014 \bbl@toglobal\bbl@savetoday
3015 \bbl@toglobal\bbl@savestate
3016 \bbl@savestrings
3017 \fi
3018 \fi}

```

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

```

3019 \def\bbl@inikv#1#2{% key=value
3020 \toks@{#2}% This hides #'s from ini values
3021 \bbl@csarg\xdef{@kv@{bbl@section.#1}{\the\toks@}}

```

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

```

3022 \let\bbl@inikv@identification\bbl@inikv
3023 \let\bbl@inikv@date\bbl@inikv
3024 \let\bbl@inikv@typography\bbl@inikv
3025 \let\bbl@inikv@characters\bbl@inikv
3026 \let\bbl@inikv@numbers\bbl@inikv

```

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

```

3027 \def\bbl@inikv@counters#1#2{%
3028 \bbl@ifsamestring{#1}{digits}%
3029 { \bbl@error{The counter name 'digits' is reserved for mapping\\
3030 decimal digits}%
3031 {Use another name.}}%
3032 }%
3033 \def\bbl@tempc{#1}%
3034 \bbl@trim@def{\bbl@tempb*}{#2}%
3035 \in@{.1$}{#1$}%
3036 \ifin@
3037 \bbl@replace\bbl@tempc{.1}{}%
3038 \bbl@csarg\protected\xdef{cntr@{bbl@tempc @\languagename}}{%
3039 \noexpand\bbl@alphanumeric{\bbl@tempb}}%
3040 \fi
3041 \in@{.F.}{#1}%
3042 \ifin@ \else \in@{.S.}{#1} \fi
3043 \ifin@
3044 \bbl@csarg\protected\xdef{cntr@#1@\languagename}{\bbl@tempb*}%
3045 \else
3046 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa

```

```

3047 \expandafter\bb1@buildifcase\bb1@tempb* \ \ % Space after \ \
3048 \bb1@csarg{\global\expandafter\let}{\cnt@#1@\language\bb1@tempa
3049 \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:

```

3050 \ifcase\bb1@engine
3051 \bb1@csarg\def{inikv@captions.licr}#1#2{%
3052 \bb1@ini@captions@aux{#1}{#2}}
3053 \else
3054 \def\bb1@inikv@captions#1#2{%
3055 \bb1@ini@captions@aux{#1}{#2}}
3056 \fi

```

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

```

3057 \def\bb1@ini@captions@template#1#2{% string language tempa=capt-name
3058 \bb1@replace\bb1@tempa{.template}{}}%
3059 \def\bb1@toreplace{#1}{}%
3060 \bb1@replace\bb1@toreplace{[ ]}{\nobreakspace{}}%
3061 \bb1@replace\bb1@toreplace{[ ]}{\csname}%
3062 \bb1@replace\bb1@toreplace{[ ]}{\csname the}%
3063 \bb1@replace\bb1@toreplace{[ ]}{\name\endcsname{}}%
3064 \bb1@replace\bb1@toreplace{[ ]}{\endcsname{}}%
3065 \bb1@xin@{,\bb1@tempa,}{,chapter,appendix,part,}%
3066 \ifin@
3067 \@nameuse{\bb1@patch\bb1@tempa}%
3068 \global\bb1@csarg\let{\bb1@tempa fmt@#2}\bb1@toreplace
3069 \fi
3070 \bb1@xin@{,\bb1@tempa,}{,figure,table,}%
3071 \ifin@
3072 \global\bb1@csarg\let{\bb1@tempa fmt@#2}\bb1@toreplace
3073 \bb1@exp{\gdef\<fnum@\bb1@tempa>{%
3074 \\\bb1@ifunset{\bb1@bb1@tempa fmt@\language}%
3075 {\[fnum@\bb1@tempa]}%
3076 {\\\@nameuse{\bb1@bb1@tempa fmt@\language}}}%
3077 \fi}
3078 \def\bb1@ini@captions@aux#1#2{%
3079 \bb1@trim\def\bb1@tempa{#1}%
3080 \bb1@xin@{.template}{\bb1@tempa}%
3081 \ifin@
3082 \bb1@ini@captions@template{#2}\language
3083 \else
3084 \bb1@ifblank{#2}%
3085 {\bb1@exp{%
3086 \toks@{\\\bb1@nocaption{\bb1@tempa}{\language\bb1@tempa name}}}%
3087 {\bb1@trim\toks@{#2}}%
3088 \bb1@exp{%
3089 \\\bb1@add\\bb1@savestrings{%
3090 \\\SetString\<\bb1@tempa name>{\the\toks@}}}%
3091 \toks@\expandafter{\bb1@captionslist}%
3092 \bb1@exp{\in@{\<\bb1@tempa name>}{\the\toks@}}%
3093 \ifin@\else
3094 \bb1@exp{%
3095 \\\bb1@add\<\bb1@extracaps@\language>{\<\bb1@tempa name>}%
3096 \\\bb1@to\global\<\bb1@extracaps@\language>}%
3097 \fi
3098 \fi}

```

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

```

3099 \def\bb1@list@the{%
3100 part,chapter,section,subsection,subsubsection,paragraph,%
3101 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3102 table,page,footnote,mpfootnote,mpfn}

```

```

3103 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3104 \bbl@ifunset{\bbl@map@#1\languagename}%
3105 {\@nameuse{#1}}%
3106 {\@nameuse{\bbl@map@#1\languagename}}}
3107 \def\bbl@inikv@labels#1#2{%
3108 \in@{.map}{#1}%
3109 \ifin@
3110 \ifx\bbl@KVP@labels\@nnil\else
3111 \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3112 \ifin@
3113 \def\bbl@tempc{#1}%
3114 \bbl@replace\bbl@tempc{.map}{}%
3115 \in@{, #2, }{, arabic, roman, Roman, alph, Alph, fnsymbol,}%
3116 \bbl@exp{%
3117 \gdef<\bbl@map@\bbl@tempc @\languagename>%
3118 {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3119 \bbl@foreach\bbl@list@the{%
3120 \bbl@ifunset{the##1}{}%
3121 {\bbl@exp{\let\\bbl@tempd<the##1>%
3122 \bbl@exp{%
3123 \\bbl@sreplace<the##1>%
3124 {\<\bbl@tempc>{##1}}{\bbl@map@cnt{\bbl@tempc}{##1}}%
3125 \\bbl@sreplace<the##1>%
3126 {\<\empty @\bbl@tempc>\<c@##1>}{\bbl@map@cnt{\bbl@tempc}{##1}}}%
3127 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3128 \toks@ \expandafter\expandafter\expandafter{%
3129 \csname the##1\endcsname}%
3130 \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3131 \fi}}%
3132 \fi
3133 \fi
3134 %
3135 \else
3136 %
3137 % The following code is still under study. You can test it and make
3138 % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3139 % language dependent.
3140 \in@{enumerate.}{#1}%
3141 \ifin@
3142 \def\bbl@tempa{#1}%
3143 \bbl@replace\bbl@tempa{enumerate.}{}%
3144 \def\bbl@toreplace{#2}%
3145 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3146 \bbl@replace\bbl@toreplace{[]}{\csname the}%
3147 \bbl@replace\bbl@toreplace{[]}{\endcsname}}%
3148 \toks@ \expandafter{\bbl@toreplace}%
3149 % TODO. Execute only once:
3150 \bbl@exp{%
3151 \\bbl@add<extras\languagename>{%
3152 \\babel@save<labelenum\romannumeral\bbl@tempa>%
3153 \def<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3154 \\bbl@tglobal<extras\languagename>}%
3155 \fi
3156 \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.

```

3157 \def\bbl@chapttype{chapter}
3158 \ifx\@makechapterhead\undefined
3159 \let\bbl@patchchapter\relax
3160 \else\ifx\thechapter\undefined

```

```

3161 \let\bbl@patchchapter\relax
3162 \else\ifx\ps@headings\undefined
3163 \let\bbl@patchchapter\relax
3164 \else
3165 \def\bbl@patchchapter{%
3166 \global\let\bbl@patchchapter\relax
3167 \gdef\bbl@chfmt{%
3168 \bbl@ifunset{bbl@bbl@chapttype fmt@\language}%
3169 {\@chapapp\space\thechapter}
3170 {\@nameuse{bbl@bbl@chapttype fmt@\language}}}
3171 \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3172 \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3173 \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3174 \bbl@sreplace\makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3175 \bbl@tglobal\appendix
3176 \bbl@tglobal\ps@headings
3177 \bbl@tglobal\chaptermark
3178 \bbl@tglobal\makechapterhead}
3179 \let\bbl@patchappendix\bbl@patchchapter
3180 \fi\fi\fi
3181 \ifx\@part\undefined
3182 \let\bbl@patchpart\relax
3183 \else
3184 \def\bbl@patchpart{%
3185 \global\let\bbl@patchpart\relax
3186 \gdef\bbl@partformat{%
3187 \bbl@ifunset{bbl@partfmt@\language}%
3188 {\partname\nobreakspace\thepart}
3189 {\@nameuse{bbl@partfmt@\language}}}
3190 \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3191 \bbl@tglobal\@part}
3192 \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

```

3193 \let\bbl@calendar\@empty
3194 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3195 \def\bbl@localedate#1#2#3#4{%
3196 \begingroup
3197 \edef\bbl@they{#2}%
3198 \edef\bbl@them{#3}%
3199 \edef\bbl@thed{#4}%
3200 \edef\bbl@tempe{%
3201 \bbl@ifunset{bbl@calpr@\language}{\bbl@cl{calpr}},%
3202 #1}%
3203 \bbl@replace\bbl@tempe{ }{}%
3204 \bbl@replace\bbl@tempe{CONVERT}{convert=}% Hackish
3205 \bbl@replace\bbl@tempe{convert}{convert=}%
3206 \let\bbl@ld@calendar\@empty
3207 \let\bbl@ld@variant\@empty
3208 \let\bbl@ld@convert\relax
3209 \def\bbl@tempb##1=##2\@{\@namedef{bbl@ld@##1}{##2}}%
3210 \bbl@foreach\bbl@tempe{\bbl@tempb##1\@}%
3211 \bbl@replace\bbl@ld@calendar{gregorian}{}%
3212 \ifx\bbl@ld@calendar\@empty\else
3213 \ifx\bbl@ld@convert\relax\else
3214 \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3215 {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3216 \fi
3217 \fi
3218 \@nameuse{bbl@precalendar}% Remove, eg, +, -civil (-ca-islamic)
3219 \edef\bbl@calendar{% Used in \month..., too
3220 \bbl@ld@calendar

```

```

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

```

**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).

```

3258 \let\bbl@calendar\@empty
3259 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3260 \@nameuse{\bbl@ca@#2}#1@@}
3261 \newcommand\babelDateSpace{\nobreakspace}
3262 \newcommand\babelDateDot{.\@} % TODO. \let instead of repeating
3263 \newcommand\babelDated[1]{\number#1}
3264 \newcommand\babelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3265 \newcommand\babelDateM[1]{\number#1}
3266 \newcommand\babelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3267 \newcommand\babelDateMMMM[1]{%
3268 \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3269 \newcommand\babelDatey[1]{\number#1}%
3270 \newcommand\babelDateyy[1]{%
3271 \ifnum#1<10 0\number#1 %
3272 \else\ifnum#1<100 \number#1 %
3273 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3274 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3275 \else
3276 \bbl@error
3277 {Currently two-digit years are restricted to the\

```



```

3278         range 0-9999.}%
3279     {There is little you can do. Sorry.}%
3280     \fi\fi\fi\fi}}
3281 \newcommand\BabelDateyyyy[1]{\{number#1\} % TODO - add leading 0
3282 \def\bbl@replace@finish@iii#1{%
3283     \bbl@exp{\def\#1####1####2####3\the\toks@}}
3284 \def\bbl@TG@date{%
3285     \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3286     \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3287     \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3288     \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3289     \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3290     \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3291     \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMM{####2}}%
3292     \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3293     \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3294     \bbl@replace\bbl@toreplace{[yyy]}{\BabelDateyyy{####1}}%
3295     \bbl@replace\bbl@toreplace{[y]}{\bbl@datecctr[####1]}%
3296     \bbl@replace\bbl@toreplace{[m]}{\bbl@datecctr[####2]}%
3297     \bbl@replace\bbl@toreplace{[d]}{\bbl@datecctr[####3]}%
3298     \bbl@replace@finish@iii\bbl@toreplace}
3299 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3300 \def\bbl@xdatecctr[#1|#2]{\localenumeral{#2}{#1}}

```

#### Transforms.

```

3301 \let\bbl@release@transforms\empty
3302 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3303 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3304 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3305     #1[#2]{#3}{#4}{#5}}
3306 \begingroup % A hack. TODO. Don't require an specific order
3307     \catcode`\%=12
3308     \catcode`\&=14
3309     \gdef\bbl@transforms#1#2#3{&%
3310         \directlua{
3311             local str = [=[#2]=]
3312             str = str:gsub('%.%d+%.%d+$', '')
3313             token.set_macro('babeltempa', str)
3314         }&%
3315         \def\babeltempc{&%
3316             \bbl@xin@{,\babeltempa,},{,\bbl@KVP@transforms,}&%
3317             \ifin@ \else
3318                 \bbl@xin@{: \babeltempa,},{,\bbl@KVP@transforms,}&%
3319             \fi
3320             \ifin@
3321                 \bbl@foreach\bbl@KVP@transforms{&%
3322                     \bbl@xin@{: \babeltempa,},{,##1,}&%
3323                     \ifin@ &% font:font:transform syntax
3324                     \directlua{
3325                         local t = {}
3326                         for m in string.gmatch('##1'..' ':'', '(-):') do
3327                             table.insert(t, m)
3328                         end
3329                         table.remove(t)
3330                         token.set_macro('babeltempc', ' ,fonts=' .. table.concat(t, ' '))
3331                     }&%
3332                     \fi}&%
3333             \in@{.0$}{#2$}&%
3334             \ifin@
3335                 \directlua{&% (\attribute) syntax
3336                     local str = string.match([[ \bbl@KVP@transforms]],
3337                         '%(([^%(-)]^%)[^%)]-\babeltempa')
3338                     if str == nil then

```

```

3339         token.set_macro('babeltempb', '')
3340     else
3341         token.set_macro('babeltempb', ',attribute=' .. str)
3342     end
3343 }&%
3344 \toks@{#3}&%
3345 \bbl@exp{&%
3346     \\g@addto@macro\\bbl@release@transforms{&%
3347         \relax &% Closes previous \bbl@transforms@aux
3348         \\bbl@transforms@aux
3349         \\#1{label=\babeltempa\babeltempb\babeltempc}&%
3350         {\languagename}{\the\toks@}}&%
3351 \else
3352     \g@addto@macro\bbl@release@transforms{, {#3}}&%
3353 \fi
3354 \fi}
3355 \endgroup

```

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

```

3356 \def\bbl@provide@lsys#1{%
3357     \bbl@ifunset{bbl@lname@#1}%
3358     {\bbl@load@info{#1}}%
3359     }%
3360     \bbl@csarg\let{lsys@#1}\@empty
3361     \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3362     \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3363     \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3364     \bbl@ifunset{bbl@lname@#1}{}%
3365     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3366     \ifcase\bbl@engine\or\or
3367         \bbl@ifunset{bbl@prehc@#1}{}%
3368         {\bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3369         }%
3370         {\ifx\bbl@xenoxyph\@undefined
3371             \global\let\bbl@xenoxyph\bbl@xenoxyph@d
3372             \ifx\AtBeginDocument\@notprerr
3373                 \expandafter\@secondoftwo % to execute right now
3374             \fi
3375             \AtBeginDocument{%
3376                 \bbl@patchfont{\bbl@xenoxyph}%
3377                 \expandafter\select@language\expandafter{\languagename}}%
3378             \fi}}%
3379     \fi
3380     \bbl@csarg\bbl@to@global{lsys@#1}}
3381 \def\bbl@xenoxyph@d{%
3382     \bbl@ifset{bbl@prehc@\languagename}%
3383     {\ifnum\hyphenchar\font=\defaultshyphenchar
3384         \iffontchar\font\bbl@cl{prehc}\relax
3385         \hyphenchar\font\bbl@cl{prehc}\relax
3386     \else\iffontchar\font"200B
3387         \hyphenchar\font"200B
3388     \else
3389         \bbl@warning
3390         {Neither 0 nor ZERO WIDTH SPACE are available\\%
3391         in the current font, and therefore the hyphen\\%
3392         will be printed. Try changing the fontspec's\\%
3393         'HyphenChar' to another value, but be aware\\%
3394         this setting is not safe (see the manual).\\%
3395         Reported}%
3396         \hyphenchar\font\defaultshyphenchar
3397     \fi\fi
3398     \fi}%

```

```
3399 {\hyphenchar\font\defaultthyphenchar}}
3400 % \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).

```

3401 \def\bbl@load@info#1{%
3402   \def\BabelBeforeIni##1##2{%
3403     \begingroup
3404       \bbl@read@ini{##1}0%
3405       \endinput           % babel- .tex may contain onlypreamble's
3406     \endgroup}%          boxed, to avoid extra spaces:
3407   {\bbl@input@texini{##1}}}
```

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

[illegible]

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

```

3439 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}%
3440   \ifx\\#1%           % \\ before, in case #1 is multiletter
3441     \bbl@exp{%
3442       \def\\bbl@tempa####1{%
3443         \<ifcase>####1\space\the\toks@\<else>\\@ctrerrr\<fi>}}%
3444     \else
3445       \toks@\expandafter{\the\toks@\or #1}%
3446       \expandafter\bbl@buildifcase
3447   \fi}

```

The code for additive counters is somewhat tricky and it's based on the fact the arguments just before \@@ collects digits which have been left 'unused' in previous arguments, the first of them being the number of digits in the number to be converted. This explains the reverse set 76543210.

Digits above 10000 are not handled yet. When the key contains the subkey .F., the number after is treated as an special case, for a fixed form (see babel-he.ini, for example).

```

3448 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@\language}\{#2}}
3449 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3450 \newcommand\localecounter[2]{%
3451   \expandafter\bbl@localecntr
3452   \expandafter\{number\csname c@#2\endcsname}\{#1}}
3453 \def\bbl@alphanumeric#1#2{%
3454   \expandafter\bbl@alphanumeric@i\number#2 76543210\@@{#1}}
3455 \def\bbl@alphanumeric@i#1#2#3#4#5#6#7#8\@#9{%
3456   \ifcase\car#8\@nil\or   % Currenty <10000, but prepared for bigger
3457     \bbl@alphanumeric@ii{#9}00000#1\or
3458     \bbl@alphanumeric@ii{#9}00000#1#2\or
3459     \bbl@alphanumeric@ii{#9}00000#1#2#3\or
3460     \bbl@alphanumeric@ii{#9}000#1#2#3#4\else
3461     \bbl@alphanumeric@invalid{>9999}%
3462   \fi}
3463 \def\bbl@alphanumeric@ii#1#2#3#4#5#6#7#8{%
3464   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\language}%
3465     {\bbl@cs{cntr@#1.4@\language}\{#5}}
3466     {\bbl@cs{cntr@#1.3@\language}\{#6}}
3467     {\bbl@cs{cntr@#1.2@\language}\{#7}}
3468     {\bbl@cs{cntr@#1.1@\language}\{#8}}
3469     \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3470     \bbl@ifunset{bbl@cntr@#1.S.321@\language}\{#%
3471       {\bbl@cs{cntr@#1.S.321@\language}\{#%
3472     \fi}%
3473     {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}\{#%
3474 \def\bbl@alphanumeric@invalid#1{%
3475   \bbl@error{Alphabetic numeral too large (#1)}%
3476   {Currently this is the limit.}}

```

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

```

3477 \def\bbl@localeinfo#1#2{%
3478   \bbl@ifunset{bbl@info@#2}\{#1}%
3479   {\bbl@ifunset{bbl@csname bbl@info@#2\endcsname @\language}\{#1}%
3480     {\bbl@cs{csname bbl@info@#2\endcsname @\language}\{#1}}
3481 \newcommand\localeinfo[1]{%
3482   \ifx*#1\@empty % TODO. A bit hackish to make it expandable.
3483     \bbl@afterelse\bbl@localeinfo{}%
3484   \else
3485     \bbl@localeinfo
3486     {\bbl@error{I've found no info for the current locale.\%
3487       The corresponding ini file has not been loaded\%
3488       Perhaps it doesn't exist}%
3489     {See the manual for details.}}%
3490     {#1}%
3491   \fi}
3492 % \@namedef{bbl@info@name.locale}\{lcname}
3493 \@namedef{bbl@info@tag.ini}\{lini}
3494 \@namedef{bbl@info@name.english}\{elname}
3495 \@namedef{bbl@info@name.opentype}\{lname}
3496 \@namedef{bbl@info@tag.bcp47}\{tbc}
3497 \@namedef{bbl@info@language.tag.bcp47}\{lbc}
3498 \@namedef{bbl@info@tag.opentype}\{lotf}
3499 \@namedef{bbl@info@script.name}\{esname}
3500 \@namedef{bbl@info@script.name.opentype}\{sname}
3501 \@namedef{bbl@info@script.tag.bcp47}\{sbcp}
3502 \@namedef{bbl@info@script.tag.opentype}\{sotf}
3503 \@namedef{bbl@info@region.tag.bcp47}\{rbcp}
3504 \@namedef{bbl@info@variant.tag.bcp47}\{vbcp}
3505 \@namedef{bbl@info@extension.t.tag.bcp47}\{extt}

```

```

3506 \@namedef{bbl@info@extension.u.tag.bcp47}{extu}
3507 \@namedef{bbl@info@extension.x.tag.bcp47}{extx}

```

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

```

3508 \providecommand\BCPdata{}
3509 \ifx\renewcommand\@undefined\else % For plain. TODO. It's a quick fix
3510   \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty}
3511   \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3512     \nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3513     {\bbl@bcpdata@ii{#6}\bbl@main@language}%
3514     {\bbl@bcpdata@ii{#1#2#3#4#5#6}\language}%
3515   \def\bbl@bcpdata@ii#1#2{%
3516     \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3517     {\bbl@error{Unknown field '#1' in \string\BCPdata.\%
3518       Perhaps you misspelled it.}%
3519     {See the manual for details.}}%
3520     {\bbl@ifunset{bbl@csname bbl@info@#1.tag.bcp47\endcsname @#2}{}%
3521     {\bbl@cs{csname bbl@info@#1.tag.bcp47\endcsname @#2}}}%
3522 \fi
3523 % Still somewhat hackish:
3524 \@namedef{bbl@info@casing.tag.bcp47}{casing}

```

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

```

3525 <<(*More package options)> \equiv
3526 \DeclareOption{ensureinfo=off}{}
3527 <</More package options>
3528 \let\bbl@ensureinfo\@gobble
3529 \newcommand\BabelEnsureInfo{%
3530   \ifx\InputIfFileExists\@undefined\else
3531     \def\bbl@ensureinfo##1{%
3532       \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3533   \fi
3534   \bbl@foreach\bbl@loaded{%
3535     \let\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files
3536     \def\language{##1}%
3537     \bbl@ensureinfo{##1}}}%
3538 \@ifpackagewith{babel}{ensureinfo=off}{}%
3539 {\AtEndOfPackage{% Test for plain.
3540   \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`.

```

3541 \newcommand\getlocaleproperty{%
3542   \ifstar\bbl@getproperty@s\bbl@getproperty@x}
3543 \def\bbl@getproperty@s#1#2#3{%
3544   \let#1\relax
3545   \def\bbl@elt##1##2##3{%
3546     \bbl@ifsamestring{##1/##2}{#3}%
3547     {\providecommand#1{##3}%
3548     \def\bbl@elt###1####2####3{}}}%
3549   {}}%
3550   \bbl@cs{inidata@#2}}%
3551 \def\bbl@getproperty@x#1#2#3{%
3552   \bbl@getproperty@s{#1}{#2}{#3}%
3553   \ifx#1\relax
3554     \bbl@error
3555     {Unknown key for locale '#2':\%
3556     #3\%
3557     \string#1 will be set to \relax}%
3558   {Perhaps you misspelled it.}%
3559   \fi}

```

```

3560 \let\bbl@ini@loaded@empty
3561 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

## 5 Adjusting the Babel bahavior

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

```

3562 \newcommand\babeladjust[1]{% TODO. Error handling.
3563   \bbl@forkv{#1}{%
3564     \bbl@ifunset{\bbl@ADJ@##1@##2}%
3565     {\bbl@cs{ADJ@##1}{##2}}%
3566     {\bbl@cs{ADJ@##1@##2}}}
3567 %
3568 \def\bbl@adjust@lua#1#2{%
3569   \ifvmode
3570     \ifnum\currentgrouplevel=\z@
3571       \directlua{ Babel.#2 }%
3572       \expandafter\expandafter\expandafter@gobble
3573     \fi
3574   \fi
3575   {\bbl@error % The error is gobbled if everything went ok.
3576     {Currently, #1 related features can be adjusted only\\%
3577       in the main vertical list.}%
3578     {Maybe things change in the future, but this is what it is.}}}
3579 \@namedef{\bbl@ADJ@bidi.mirroring@on}{%
3580   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3581 \@namedef{\bbl@ADJ@bidi.mirroring@off}{%
3582   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3583 \@namedef{\bbl@ADJ@bidi.text@on}{%
3584   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3585 \@namedef{\bbl@ADJ@bidi.text@off}{%
3586   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3587 \@namedef{\bbl@ADJ@bidi.math@on}{%
3588   \let\bbl@noamsmath\empty}
3589 \@namedef{\bbl@ADJ@bidi.math@off}{%
3590   \let\bbl@noamsmath\relax}
3591 \@namedef{\bbl@ADJ@bidi.mapdigits@on}{%
3592   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3593 \@namedef{\bbl@ADJ@bidi.mapdigits@off}{%
3594   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3595 %
3596 \@namedef{\bbl@ADJ@linebreak.sea@on}{%
3597   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3598 \@namedef{\bbl@ADJ@linebreak.sea@off}{%
3599   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3600 \@namedef{\bbl@ADJ@linebreak.cjk@on}{%
3601   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3602 \@namedef{\bbl@ADJ@linebreak.cjk@off}{%
3603   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3604 \@namedef{\bbl@ADJ@justify.arabic@on}{%
3605   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3606 \@namedef{\bbl@ADJ@justify.arabic@off}{%
3607   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3608 %
3609 \def\bbl@adjust@layout#1{%
3610   \ifvmode
3611     #1%
3612     \expandafter@gobble
3613   \fi
3614   {\bbl@error % The error is gobbled if everything went ok.
3615     {Currently, layout related features can be adjusted only\\%
3616       in vertical mode.}%
3617     {Maybe things change in the future, but this is what it is.}}}

```

```

3618 \@namedef{bbl@ADJ@layout.tabular@on}{%
3619   \ifnum\bbl@tabular@mode=\tw@
3620     \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}%
3621   \else
3622     \chardef\bbl@tabular@mode\@ne
3623   \fi}
3624 \@namedef{bbl@ADJ@layout.tabular@off}{%
3625   \ifnum\bbl@tabular@mode=\tw@
3626     \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}%
3627   \else
3628     \chardef\bbl@tabular@mode\z@
3629   \fi}
3630 \@namedef{bbl@ADJ@layout.lists@on}{%
3631   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3632 \@namedef{bbl@ADJ@layout.lists@off}{%
3633   \bbl@adjust@layout{\let\list\bbl@OL@list}}
3634 %
3635 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3636   \bbl@bcpallowedtrue}
3637 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3638   \bbl@bcpallowedfalse}
3639 \@namedef{bbl@ADJ@autoload.bcp47.prefix{#1}}{
3640   \def\bbl@bcp@prefix{#1}}
3641 \def\bbl@bcp@prefix{bcp47-}
3642 \@namedef{bbl@ADJ@autoload.options{#1}}{
3643   \def\bbl@autoload@options{#1}}
3644 \let\bbl@autoload@bcptoptions\empty
3645 \@namedef{bbl@ADJ@autoload.bcp47.options{#1}}{
3646   \def\bbl@autoload@bcptoptions{#1}}
3647 \newif\ifbbl@bcptoname
3648 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3649   \bbl@bcptonametrue
3650   \BabelEnsureInfo}
3651 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3652   \bbl@bcptonamefalse}
3653 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3654   \directlua{ Babel.ignore_pre_char = function(node)
3655     return (node.lang == \the\csname l@nohyphenation\endcsname)
3656   end }}
3657 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3658   \directlua{ Babel.ignore_pre_char = function(node)
3659     return false
3660   end }}
3661 \@namedef{bbl@ADJ@select.write@shift}{%
3662   \let\bbl@restorelastskip\relax
3663   \def\bbl@savelastskip{%
3664     \let\bbl@restorelastskip\relax
3665     \ifvmode
3666       \ifdim\lastskip=\z@
3667         \let\bbl@restorelastskip\nobreak
3668       \else
3669         \bbl@exp{%
3670           \def\\bbl@restorelastskip{%
3671             \skip@=\the\lastskip
3672             \\nobreak \vskip-\skip@ \vskip\skip@}}%
3673         \fi
3674       \fi}}
3675 \@namedef{bbl@ADJ@select.write@keep}{%
3676   \let\bbl@restorelastskip\relax
3677   \let\bbl@savelastskip\relax}
3678 \@namedef{bbl@ADJ@select.write@omit}{%
3679   \AddBabelHook{babel-select}{beforestart}{%
3680     \expandafter\babel@aux\expandafter{\bbl@main@language}}}%

```

```

3681 \let\bbl@restorelastskip\relax
3682 \def\bbl@savelastskip##1\bbl@restorelastskip{}}
3683 \@namedef{bbl@ADJ@select.encoding@off}{%
3684 \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.

```

3685 <<More package options>> ≡
3686 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3687 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3688 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3689 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3690 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3691 <</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.

```

3692 \bbl@trace{Cross referencing macros}
3693 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3694 \def\@newl@bel#1#2#3{%
3695   {\@safe@activestrue
3696     \bbl@ifunset{#1#2}%
3697       \relax
3698       {\gdef\@multiplelabels{%
3699         \@latex@warning@no@line{There were multiply-defined labels}}%
3700         \@latex@warning@no@line{Label `#2' multiply defined}}%
3701       \global\@namedef{#1#2}{#3}}}%

```

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

```

3702 \CheckCommand*\@testdef[3]{%
3703   \def\reserved@a{#3}%
3704   \expandafter\ifx\csname#1#2\endcsname\reserved@a
3705   \else
3706     \@tempswatrue
3707   \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.

```

3708 \def\@testdef#1#2#3{% TODO. With @samestring?
3709   \@safe@activestrue
3710   \expandafter\let\expandafter\bbl@tempa\csname #1#2\endcsname
3711   \def\bbl@tempb{#3}%
3712   \@safe@activesfalse
3713   \ifx\bbl@tempa\relax
3714   \else
3715     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3716   \fi
3717   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%

```



```

3718 \ifx\bbl@tempa\bbl@tempb
3719 \else
3720 \@tempswattrue
3721 \fi}
3722 \fi

```

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

```

3723 \bbl@xin@{R}\bbl@opt@safe
3724 \ifin@
3725 \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3726 \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3727 {\expandafter\strip@prefix\meaning\ref}%
3728 \ifin@
3729 \bbl@redefine\@kernel@ref#1{%
3730 \@safe@activetrue\org@@kernel@ref{#1}\@safe@activesfalse}
3731 \bbl@redefine\@kernel@pageref#1{%
3732 \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activesfalse}
3733 \bbl@redefine\@kernel@sref#1{%
3734 \@safe@activetrue\org@@kernel@sref{#1}\@safe@activesfalse}
3735 \bbl@redefine\@kernel@spageref#1{%
3736 \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activesfalse}
3737 \else
3738 \bbl@redefinero bust\ref#1{%
3739 \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
3740 \bbl@redefinero bust\pageref#1{%
3741 \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
3742 \fi
3743 \else
3744 \let\org@ref\ref
3745 \let\org@pageref\pageref
3746 \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.

```

3747 \bbl@xin@{B}\bbl@opt@safe
3748 \ifin@
3749 \bbl@redefine\@citex[#1]#2{%
3750 \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
3751 \org@@citex[#1]{\@tempa}}

```

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

```

3752 \AtBeginDocument{%
3753 \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.)

```

3754 \def\@citex[#1][#2]#3{%
3755 \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
3756 \org@@citex[#1][#2]{\@tempa}}%
3757 }{}

```

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

```

3758 \AtBeginDocument{%

```

```

3759 \ifpackageloaded{cite}{%
3760 \def\@citex[#1]#2{%
3761 \@safe@activetrue\org@citex[#1]#2}\@safe@activesfalse}%
3762 }{}}

```

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

```

3763 \bbl@redefine\nocite#1{%
3764 \@safe@activetrue\org@nocite{#1}\@safe@activesfalse}

```

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

```

3765 \bbl@redefine\bibcite{%
3766 \bbl@cite@choice
3767 \bibcite}

```

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

```

3768 \def\bbl@bibcite#1#2{%
3769 \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.

```

3770 \def\bbl@cite@choice{%
3771 \global\let\bibcite\bbl@bibcite
3772 \ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
3773 \ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
3774 \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.

```

3775 \AtBeginDocument{\bbl@cite@choice}

```

`\@bibitem` One of the two internal L<sup>A</sup>T<sub>E</sub>X macros called by `\bibitem` that write the citation label on the .aux file.

```

3776 \bbl@redefine\@bibitem#1{%
3777 \@safe@activetrue\org@@bibitem{#1}\@safe@activesfalse}
3778 \else
3779 \let\org@nocite\nocite
3780 \let\org@@citex\@citex
3781 \let\org@bibcite\bibcite
3782 \let\org@@bibitem\@bibitem
3783 \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.

```

3784 \bbl@trace{Marks}
3785 \IfBabelLayout{sectioning}
3786 {\ifx\bbl@opt@headfoot\@nnil
3787 \g@addto@macro\@resetactivechars{%
3788 \set@typeset@protect
3789 \expandafter\select@language\x\expandafter{\bbl@main@language}%
3790 \let\protect\noexpand

```

```

3791 \ifcase\bbl@bidimode\else % Only with bidi. See also above
3792 \edef\thepage{%
3793 \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3794 \fi}%
3795 \fi}
3796 {\ifbbl@single\else
3797 \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
3798 \markright#1{%
3799 \bbl@ifblank{#1}%
3800 {\org@markright{}}}%
3801 {\toks@{#1}%
3802 \bbl@exp{%
3803 \org@markright{\protect\foreignlanguage{\language}\thepage}%
3804 {\protect\bbl@restore@actives\the\toks@}}}%

```

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

```

3805 \ifx\mkboth\markboth
3806 \def\bbl@tempc{\let\mkboth\markboth}%
3807 \else
3808 \def\bbl@tempc{}%
3809 \fi
3810 \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
3811 \markboth#1#2{%
3812 \protected@edef\bbl@tempb##1{%
3813 \protect\foreignlanguage
3814 {\language}\protect\bbl@restore@actives##1}}%
3815 \bbl@ifblank{#1}%
3816 {\toks@{}}%
3817 {\toks@\expandafter{\bbl@tempb{#1}}}%
3818 \bbl@ifblank{#2}%
3819 {\@temptokena{}}%
3820 {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3821 \bbl@exp{\org@markboth{\the\toks@}{\the\@temptokena}}}%
3822 \bbl@tempc
3823 \fi} % end ifbbl@single, end \IfBabelLayout

```

## 5.3 Preventing clashes with other packages

### 5.3.1 `ifthen`

`\ifthenelse` Sometimes a document writer wants to create a special effect depending on the page a certain fragment of text appears on. This can be achieved by the following piece of code:

```

\ifthenelse{\isodd{\pageref{some:label}}}
{code for odd pages}
{code for even pages}

```

In order for this to work the argument of `\isodd` needs to be fully expandable. With the above redefinition of `\pageref` it is not in the case of this example. To overcome that, we add some code to the definition of `\ifthenelse` to make things work.

We want to revert the definition of `\pageref` and `\ref` to their original definition for the first argument of `\ifthenelse`, so we first need to store their current meanings.

Then we can set the `\safe@actives` switch and call the original `\ifthenelse`. In order to be able to use shorthands in the second and third arguments of `\ifthenelse` the resetting of the switch *and* the definition of `\pageref` happens inside those arguments.

```

3824 \bbl@trace{Preventing clashes with other packages}
3825 \ifx\org@ref\undefined\else

```

```

3826 \bbl@xin@{R}\bbl@opt@safe
3827 \ifin@
3828 \AtBeginDocument{%
3829 \ifpackageloaded{ifthen}{%
3830 \bbl@redefine@long\ifthenelse#1#2#3{%
3831 \let\bbl@temp@pref\pageref
3832 \let\pageref\org@pageref
3833 \let\bbl@temp@ref\ref
3834 \let\ref\org@ref
3835 \@safe@activetrue
3836 \org@ifthenelse{#1}%
3837 {\let\pageref\bbl@temp@pref
3838 \let\ref\bbl@temp@ref
3839 \@safe@activetrue
3840 #2}%
3841 {\let\pageref\bbl@temp@pref
3842 \let\ref\bbl@temp@ref
3843 \@safe@activetrue
3844 #3}%
3845 }%
3846 }{}%
3847 }
3848 \fi

```

### 5.3.2 varioref

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

```

3849 \AtBeginDocument{%
3850 \ifpackageloaded{varioref}{%
3851 \bbl@redefine\@@vpageref#1[#2]#3{%
3852 \@safe@activetrue
3853 \org@@@vpageref{#1}[#2]#3}%
3854 \@safe@activetrue}%
3855 \bbl@redefine\vrefpagemum#1#2{%
3856 \@safe@activetrue
3857 \org@vrefpagemum{#1}#2}%
3858 \@safe@activetrue}%

```

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.

```

3859 \expandafter\def\csname Ref \endcsname#1{%
3860 \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3861 }{}%
3862 }
3863 \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.

```

3864 \AtEndOfPackage{%
3865 \AtBeginDocument{%
3866 \ifpackageloaded{hhline}{%
3867 {\expandafter\ifx\csname normal@char\string\endcsname\relax
3868 \else

```

```

3869         \makeatletter
3870         \def\@currname{hhline}\input{hhline.sty}\makeatother
3871         \fi}%
3872     {}}}

```

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

```

3873 \def\substitutefontfamily#1#2#3{%
3874     \lowercase{\immediate\openout15=#1#2.fd\relax}%
3875     \immediate\write15{%
3876         \string\ProvidesFile{#1#2.fd}%
3877         [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3878         \space generated font description file]^^J
3879         \string\DeclareFontFamily{#1}{#2}{^^J
3880         \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
3881         \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
3882         \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
3883         \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
3884         \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
3885         \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
3886         \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
3887         \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
3888     }%
3889     \closeout15
3890 }
3891 \@onlypreamble\substitutefontfamily

```

## 5.4 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of  $\text{\TeX}$  and  $\text{\LaTeX}$  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

3892 \bbl@trace{Encoding and fonts}
3893 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3894 \newcommand\BabelNonText{TS1,T3,TS3}
3895 \let\org@TeX\TeX
3896 \let\org@LaTeX\LaTeX
3897 \let\ensureascii@firstofone
3898 \AtBeginDocument{%
3899     \def\@elt#1{,#1,}%
3900     \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3901     \let\@elt\relax
3902     \let\bbl@tempb\@empty
3903     \def\bbl@tempc{OT1}%
3904     \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3905         \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3906     \bbl@foreach\bbl@tempa{%
3907         \bbl@xin@{#1}{\BabelNonASCII}%
3908         \ifin@
3909             \def\bbl@tempb{#1}% Store last non-ascii
3910         \else\bbl@xin@{#1}{\BabelNonText}% Pass
3911             \ifin@\else
3912                 \def\bbl@tempc{#1}% Store last ascii
3913             \fi
3914         \fi}%
3915     \ifx\bbl@tempb\@empty\else
3916         \bbl@xin@{\cf@encoding,}{\BabelNonASCII,\BabelNonText,}%
3917         \ifin@\else

```

```

3918     \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3919     \fi
3920     \edef\ensureascii#1{%
3921       {\noexpand\fontencoding{\bbl@tempc}\noexpand\selectfont#1}}%
3922     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3923     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3924     \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.

```

3925 \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.

```

3926 \AtBeginDocument{%
3927   \@ifpackageloaded{fontspec}%
3928   {\xdef\latinencoding{%
3929     \ifx\UTFencname\@undefined
3930       EU\ifcase\bbl@engine\or2\or1\fi
3931     \else
3932       \UTFencname
3933     \fi}}%
3934   {\gdef\latinencoding{OT1}%
3935     \ifx\cf@encoding\bbl@t@one
3936       \xdef\latinencoding{\bbl@t@one}%
3937     \else
3938       \def\@elt#1{,#1,}%
3939       \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3940       \let\@elt\relax
3941       \bbl@xin@{,T1,}\bbl@tempa
3942       \ifin@
3943         \xdef\latinencoding{\bbl@t@one}%
3944       \fi
3945     \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.

```

3946 \DeclareRobustCommand{\latintext}{%
3947   \fontencoding{\latinencoding}\selectfont
3948   \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.

```

3949 \ifx\@undefined\DeclareTextFontCommand
3950   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3951 \else
3952   \DeclareTextFontCommand{\textlatin}{\latintext}
3953 \fi

```

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

```

3954 \def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}

```

## 5.5 Basic bidi support

**Work in progress.** This code is currently placed here for practical reasons. It will be moved to the correct place soon, I hope.

It is loosely based on `rlbabel.def`, but most of it has been developed from scratch. This `babel` module (by Johannes Braams and Boris Lavva) has served the purpose of typesetting R documents for two decades, and despite its flaws I think it is still a good starting point (some parts have been copied here almost verbatim), partly thanks to its simplicity. I’ve also looked at `ARABI` (by Youssef Jabri), which is compatible with `babel`.

There are two ways of modifying macros to make them “bidi”, namely, by patching the internal low-level macros (which is what I have done with lists, columns, counters, tocs, much like `rlbabel` did), and by introducing a “middle layer” just below the user interface (sectioning, footnotes).

- `pdftex` provides a minimal support for bidi text, and it must be done by hand. Vertical typesetting is not possible.
- `xetex` is somewhat better, thanks to its font engine (even if not always reliable) and a few additional tools. However, very little is done at the paragraph level. Another challenging problem is text direction does not honour  $\TeX$  grouping.
- `luatex` can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but bidi text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As `Lua $\TeX$ -ja` shows, vertical typesetting is possible, too.

```
3955 \bbl@trace{Loading basic (internal) bidi support}
3956 \ifodd\bbl@engine
3957 \else % TODO. Move to txtbabel
3958   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3959     \bbl@error
3960     {The bidi method 'basic' is available only in\\%
3961     luatex. I'll continue with 'bidi=default', so\\%
3962     expect wrong results}%
3963     {See the manual for further details.}%
3964     \let\bbl@beforeforeign\leavevmode
3965     \AtEndOfPackage{%
3966       \EnableBabelHook{babel-bidi}%
3967       \bbl@xebidipar}
3968   \fi\fi
3969   \def\bbl@loadxebidi#1{%
3970     \ifx\RTLfootnotetext\@undefined
3971       \AtEndOfPackage{%
3972         \EnableBabelHook{babel-bidi}%
3973         \bbl@loadfontspec % bidi needs fontspec
3974         \usepackage#1{bidi}}%
3975     \fi}
3976   \ifnum\bbl@bidimode>200
3977     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3978       \bbl@tentative{bidi=bidi}
3979       \bbl@loadxebidi{}
3980     \or
3981       \bbl@loadxebidi{[rldocument]}
3982     \or
3983       \bbl@loadxebidi{}
3984     \fi
3985   \fi
3986 \fi
3987 % TODO? Separate:
3988 \ifnum\bbl@bidimode=\@ne
3989   \let\bbl@beforeforeign\leavevmode
3990   \ifodd\bbl@engine
3991     \newattribute\bbl@attr@dir
3992     \directlua{ Babel.attr_dir = luatexbase.registernumber' bbl@attr@dir' }
3993     \bbl@exp{\output{\bodydir\pagedir\the\output}}
3994   \fi
3995   \AtEndOfPackage{%
```

```

3996 \EnableBabelHook{babel-bidi}%
3997 \ifodd\bbbl@engine\else
3998 \bbbl@xebidipar
3999 \fi}
4000 \fi

```

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

```

4001 \bbbl@trace{Macros to switch the text direction}
4002 \def\bbbl@alscripts{,Arabic,Syriac,Thaana,}
4003 \def\bbbl@rscripts{% TODO. Base on codes ??
4004   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
4005   Old Hungarian,Lydian,Mandaean,Manichaeen,%
4006   Meroitic Cursive,Meroitic,Old North Arabian,%
4007   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
4008   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
4009   Old South Arabian,}%
4010 \def\bbbl@provide@dirs#1{%
4011   \bbbl@xin@{\csname bbl@sname@#1\endcsname}{\bbbl@alscripts\bbbl@rscripts}%
4012   \ifin@
4013     \global\bbbl@csarg\chardef{wdir@#1}\@ne
4014     \bbbl@xin@{\csname bbl@sname@#1\endcsname}{\bbbl@alscripts}%
4015     \ifin@
4016       \global\bbbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
4017     \fi
4018   \else
4019     \global\bbbl@csarg\chardef{wdir@#1}\z@
4020   \fi
4021   \ifodd\bbbl@engine
4022     \bbbl@csarg\ifcase{wdir@#1}%
4023       \directlua{ Babel.locale_props[\the\localeid].texdir = 'l' }%
4024     \or
4025       \directlua{ Babel.locale_props[\the\localeid].texdir = 'r' }%
4026     \or
4027       \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
4028     \fi
4029   \fi}
4030 \def\bbbl@switchdir{%
4031   \bbbl@ifunset{bbl@lsys@\languagename}{\bbbl@provide@lsys{\languagename}}{}%
4032   \bbbl@ifunset{bbl@wdir@\languagename}{\bbbl@provide@dirs{\languagename}}{}%
4033   \bbbl@exp{\bbbl@setdirs\bbbl@cl{wdir}}%
4034   \def\bbbl@setdirs#1{% TODO - math
4035     \ifcase\bbbl@select@type % TODO - strictly, not the right test
4036       \bbbl@bodydir{#1}%
4037       \bbbl@pardir{#1}% <- Must precede \bbbl@texdir
4038     \fi
4039     \bbbl@texdir{#1}}
4040 % TODO. Only if \bbbl@bidimode > 0?:
4041 \AddBabelHook{babel-bidi}{afterextras}{\bbbl@switchdir}
4042 \DisableBabelHook{babel-bidi}

```

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

```

4043 \ifodd\bbbl@engine % luatex=1
4044 \else % pdftex=0, xetex=2
4045   \newcount\bbbl@dirlevel
4046   \chardef\bbbl@thetextdir\z@
4047   \chardef\bbbl@thepardir\z@
4048   \def\bbbl@texdir#1{%
4049     \ifcase#1\relax
4050       \chardef\bbbl@thetextdir\z@
4051       \bbbl@texdir@i\beginL\endL
4052     \else
4053       \chardef\bbbl@thetextdir\@ne
4054       \bbbl@texdir@i\beginR\endR

```



```

4055 \fi}
4056 \def\bbl@textdir@i#1#2{%
4057 \ifhmode
4058 \ifnum\currentgrouplevel>\z@
4059 \ifnum\currentgrouplevel=\bbl@dirlevel
4060 \bbl@error{Multiple bidi settings inside a group}%
4061 {I'll insert a new group, but expect wrong results.}%
4062 \bgroup\aftergroup#2\aftergroup\egroup
4063 \else
4064 \ifcase\currentgrouptype\or % 0 bottom
4065 \aftergroup#2% 1 simple {}
4066 \or
4067 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4068 \or
4069 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4070 \or\or\or % vbox vtop align
4071 \or
4072 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4073 \or\or\or\or\or % output math disc insert vcent mathchoice
4074 \or
4075 \aftergroup#2% 14 \begin group
4076 \else
4077 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4078 \fi
4079 \fi
4080 \bbl@dirlevel\currentgrouplevel
4081 \fi
4082 #1%
4083 \fi}
4084 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4085 \let\bbl@bodydir\@gobble
4086 \let\bbl@pagedir\@gobble
4087 \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).

```

4088 \def\bbl@xebidipar{%
4089 \let\bbl@xebidipar\relax
4090 \TeXeTstate\@ne
4091 \def\bbl@xeeverypar{%
4092 \ifcase\bbl@thepardir
4093 \ifcase\bbl@thetextdir\else\beginR\fi
4094 \else
4095 {\setbox\z@\lastbox\beginR\box\z@}%
4096 \fi}%
4097 \let\bbl@severypar\everypar
4098 \newtoks\everypar
4099 \everypar=\bbl@severypar
4100 \bbl@severypar{\bbl@xeeverypar\the\everypar}}
4101 \ifnum\bbl@bidimode>200
4102 \let\bbl@textdir@i\@gobbletwo
4103 \let\bbl@xebidipar\@empty
4104 \AddBabelHook{bidi}{foreign}{%
4105 \def\bbl@tempa{\def\BabelText###1}%
4106 \ifcase\bbl@thetextdir
4107 \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4108 \else
4109 \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4110 \fi}
4111 \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4112 \fi
4113 \fi

```

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

```

4114 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4115 \AtBeginDocument{%
4116   \ifx\pdfstringdefDisableCommands\@undefined\else
4117     \ifx\pdfstringdefDisableCommands\relax\else
4118       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4119     \fi
4120   \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`.

```

4121 \bbl@trace{Local Language Configuration}
4122 \ifx\loadlocalcfg\@undefined
4123   \ifpackagewith{babel}{noconfigs}%
4124     {\let\loadlocalcfg\@gobble}%
4125     {\def\loadlocalcfg#1{%
4126       \InputIfFileExists{#1.cfg}%
4127       {\typeout{*****^J%
4128                 * Local config file #1.cfg used^^J%
4129                 *}}%
4130       \@empty}}
4131 \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).

```

4132 \bbl@trace{Language options}
4133 \let\bbl@afterlang\relax
4134 \let\BabelModifiers\relax
4135 \let\bbl@loaded\@empty
4136 \def\bbl@load@language#1{%
4137   \InputIfFileExists{#1.ldf}%
4138   {\edef\bbl@loaded{\CurrentOption
4139     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4140     \expandafter\let\expandafter\bbl@afterlang
4141       \csname\CurrentOption.ldf-h@k\endcsname
4142     \expandafter\let\expandafter\BabelModifiers
4143       \csname\bbl@mod@\CurrentOption\endcsname
4144     \bbl@exp{\AtBeginDocument{%
4145       \bbl@usehooks@lang{\CurrentOption}{begindocument}{\CurrentOption}}}%
4146     {\bbl@error{%
4147       Unknown option '\CurrentOption'. Either you misspelled it\\%
4148       or the language definition file \CurrentOption.ldf was not found}%
4149       Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4150       activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4151       headfoot=, strings=, config=, hyphenmap=, or a language name.}}}

```

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

```

4152 \def\bbl@try@load@lang#1#2#3{%
4153   \IfFileExists{\CurrentOption.ldf}%
4154   {\bbl@load@language{\CurrentOption}}%
4155   {\bbl@load@language{#2#3}}
4156 %

```

```

4157 \DeclareOption{hebrew}{%
4158   \input{rlbabel.def}%
4159   \bbl@load@language{hebrew}}
4160 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4161 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4162 \DeclareOption{northernsami}{\bbl@try@load@lang{}{samin}{}}
4163 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4164 \DeclareOption{polutonikogreek}{%
4165   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4166 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4167 \DeclareOption{scottishgaelic}{\bbl@try@load@lang{}{scottish}{}}
4168 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4169 \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.

```

4170 \ifx\bbl@opt@config\@nnil
4171   \@ifpackagewith{babel}{noconfigs}{}%
4172   {\InputIfFileExists{bblopts.cfg}%
4173     {\typeout{*****^J%
4174               * Local config file bblopts.cfg used^^J%
4175               *}}}%
4176   {}}%
4177 \else
4178   \InputIfFileExists{\bbl@opt@config.cfg}%
4179   {\typeout{*****^J%
4180             * Local config file \bbl@opt@config.cfg used^^J%
4181             *}}%
4182   {\bbl@error{%
4183     Local config file '\bbl@opt@config.cfg' not found}{%
4184     Perhaps you misspelled it.}}%
4185 \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.

```

4186 \ifx\bbl@opt@main\@nnil
4187   \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4188     \let\bbl@tempb\@empty
4189     \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4190     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4191     \bbl@foreach\bbl@tempb{% \bbl@tempb is a reversed list
4192       \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4193         \ifodd\bbl@iniflag % = *=
4194           \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
4195         \else % n +=
4196           \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4197       \fi
4198     \fi}%
4199 \fi
4200 \else
4201   \bbl@info{Main language set with 'main='. Except if you have\\%
4202     problems, prefer the default mechanism for setting\\%
4203     the main language, ie, as the last declared.\\%
4204     Reported}%
4205 \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).

```
4206 \ifx\bbbl@opt@main\@nnil\else
4207   \bbbl@ncarg\let\bbbl@loadmain{ds@\bbbl@opt@main}%
4208   \expandafter\let\csname ds@\bbbl@opt@main\endcsname\relax
4209 \fi
```

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

```
4210 \bbbl@foreach\bbbl@language@opts{%
4211   \def\bbbl@tempa{#1}%
4212   \ifx\bbbl@tempa\bbbl@opt@main\else
4213     \ifnum\bbbl@iniflag<\tw@ % 0 0 (other = ldf)
4214       \bbbl@ifunset{ds@#1}%
4215       {\DeclareOption{#1}{\bbbl@load@language{#1}}}%
4216       {}%
4217     \else % + * (other = ini)
4218       \DeclareOption{#1}{%
4219         \bbbl@ldfinit
4220         \babelprovide[import]{#1}%
4221         \bbbl@afterldf{}}%
4222     \fi
4223   \fi}
4224 \bbbl@foreach\@classoptionslist{%
4225   \def\bbbl@tempa{#1}%
4226   \ifx\bbbl@tempa\bbbl@opt@main\else
4227     \ifnum\bbbl@iniflag<\tw@ % 0 0 (other = ldf)
4228       \bbbl@ifunset{ds@#1}%
4229       {\IfFileExists{#1.ldf}%
4230        {\DeclareOption{#1}{\bbbl@load@language{#1}}}%
4231        {}}%
4232       {}%
4233     \else % + * (other = ini)
4234       \IfFileExists{babel-#1.tex}%
4235       {\DeclareOption{#1}{%
4236         \bbbl@ldfinit
4237         \babelprovide[import]{#1}%
4238         \bbbl@afterldf{}}}%
4239       {}%
4240     \fi
4241   \fi}
```

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

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

```
4242 \def\AfterBabelLanguage#1{%
4243   \bbbl@ifsamestring\CurrentOption{#1}{\global\bbbl@add\bbbl@afterlang{}}
4244   \DeclareOption*{}
4245   \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.

```
4246 \bbbl@trace{Option 'main'}
4247 \ifx\bbbl@opt@main\@nnil
4248   \edef\bbbl@tempa{\@classoptionslist,\bbbl@language@opts}
4249   \let\bbbl@tempc\@empty
4250   \edef\bbbl@templ{\bbbl@loaded,}
4251   \edef\bbbl@templ{\expandafter\strip@prefix\meaning\bbbl@templ}
```

```

4252 \bbl@for\bbl@tempb\bbl@tempa{%
4253   \edef\bbl@tempd{\bbl@tempb,}%
4254   \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4255   \bbl@xin@{\bbl@tempd}{\bbl@templ}%
4256   \ifin@{\edef\bbl@tempc{\bbl@tempb}\fi}
4257   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4258   \expandafter\bbl@tempa\bbl@loaded,\@nnil
4259   \ifx\bbl@tempb\bbl@tempc\else
4260     \bbl@warning{%
4261       Last declared language option is '\bbl@tempc',\%
4262       but the last processed one was '\bbl@tempb'.\%
4263       The main language can't be set as both a global\%
4264       and a package option. Use 'main=\bbl@tempc' as\%
4265       option. Reported}
4266   \fi
4267 \else
4268   \ifodd\bbl@iniflag % case 1,3 (main is ini)
4269     \bbl@ldfinit
4270     \let\CurrentOption\bbl@opt@main
4271     \bbl@exp{% \bbl@opt@provide = empty if *
4272       \\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4273     \bbl@afterldf{}
4274     \DeclareOption{\bbl@opt@main}{}
4275   \else % case 0,2 (main is ldf)
4276     \ifx\bbl@loadmain\relax
4277       \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4278     \else
4279       \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4280     \fi
4281     \ExecuteOptions{\bbl@opt@main}
4282     \@namedef{ds@\bbl@opt@main}{}%
4283   \fi
4284   \DeclareOption*{}
4285   \ProcessOptions*
4286 \fi
4287 \bbl@exp{%
4288   \\AtBeginDocument{\\bbl@usehooks@lang{/}{\begin@document}{}}}%
4289 \def\AfterBabelLanguage{%
4290   \bbl@error
4291     {Too late for \string\AfterBabelLanguage}%
4292     {Languages have been loaded, so I can do nothing}}
4293 \ifx\bbl@main@language\@undefined
4294   \bbl@info{%
4295     You haven't specified a language as a class or package\%
4296     option. I'll load 'nil'. Reported}
4297   \bbl@load@language{nil}
4298 \fi
4299 </package>

```

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.

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

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

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

```
4300 <*kernel>
4301 \let\bbl@onlyswitch\@empty
4302 \input babel.def
4303 \let\bbl@onlyswitch\@undefined
4304 </kernel>
4305 <*patterns>
```

## 7 Loading hyphenation patterns

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

```
4306 <<Make sure ProvidesFile is defined>>
4307 \ProvidesFile{hyphen.cfg}[\<date> v\<version>] Babel hyphens]
4308 \xdef\bbl@format{\jobname}
4309 \def\bbl@version{\<version>}
4310 \def\bbl@date{\<date>}
4311 \ifx\AtBeginDocument\@undefined
4312   \def\@empty{}
4313 \fi
4314 <<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.

```
4315 \def\process@line#1#2 #3 #4 {%
4316   \ifx=#1%
4317     \process@synonym{#2}%
4318   \else
4319     \process@language{#1#2}{#3}{#4}%
4320   \fi
4321   \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.

```
4322 \toks@{}
4323 \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.

```
4324 \def\process@synonym#1{%
4325   \ifnum\last@language=\m@ne
4326     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4327   \else
4328     \expandafter\chardef\csname l@#1\endcsname\last@language
4329     \wlog{\string\l@#1=\string\language\the\last@language}%
4330     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4331       \csname\language\hyphenmins\endcsname
4332     \let\bbl@elt\relax
4333     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}{}%
4334   \fi}
```

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

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

For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file `language.dat` by adding for instance ‘: T1’ to the name of the language. The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

Pattern files may contain assignments to `\lefthyphenmin` and `\righthyphenmin`. T<sub>E</sub>X does not keep track of these assignments. Therefore we try to detect such assignments and store them in the `\<lang>hyphenmins` macro. When no assignments were made we provide a default setting. Some pattern files contain changes to the `\lccode` or `\uccode` arrays. Such changes should remain local to the language; therefore we process the pattern file in a group; the `\patterns` command acts globally so its effect will be remembered.

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

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

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

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

```

4335 \def\process@language#1#2#3{%
4336   \expandafter\addlanguage\csname l@#1\endcsname
4337   \expandafter\language\csname l@#1\endcsname
4338   \edef\language{#1}%
4339   \bbl@hook@everylanguage{#1}%
4340   % > luatex
4341   \bbl@get@enc#1::\@@@
4342   \begingroup
4343     \lefthyphenmin\m@ne
4344     \bbl@hook@loadpatterns{#2}%
4345     % > luatex
4346     \ifnum\lefthyphenmin=\m@ne
4347     \else
4348       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4349         \the\lefthyphenmin\the\righthyphenmin}%
4350     \fi
4351   \endgroup
4352   \def\bbl@tempa{#3}%
4353   \ifx\bbl@tempa\@empty\else
4354     \bbl@hook@loadexceptions{#3}%
4355     % > luatex
4356   \fi
4357   \let\bbl@elt\relax
4358   \edef\bbl@languages{%
4359     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4360   \ifnum\the\language=\z@
4361     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4362       \set@hyphenmins\tw@\thr@@\relax
4363     \else
4364       \expandafter\expandafter\expandafter\set@hyphenmins
4365       \csname #1hyphenmins\endcsname
4366     \fi
4367     \the\toks@
4368     \toks@{}%
4369   \fi}

```

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

```

4370 \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.

```

4371 \def\bbl@hook@everylanguage#1{}
4372 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4373 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4374 \def\bbl@hook@loadkernel#1{%
4375   \def\addlanguage{\csname newlanguage\endcsname}%
4376   \def\adddialect##1##2{%
4377     \global\chardef##1##2\relax
4378     \wlog{\string##1 = a dialect from \string\language##2}}%
4379   \def\iflanguage##1{%
4380     \expandafter\ifx\csname l@##1\endcsname\relax
4381       \@nolanerr{##1}%
4382     \else
4383       \ifnum\csname l@##1\endcsname=\language
4384         \expandafter\expandafter\expandafter\@firstoftwo
4385       \else
4386         \expandafter\expandafter\expandafter\@secondoftwo
4387       \fi
4388     \fi}%
4389   \def\providehyphenmins##1##2{%
4390     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4391       \@namedef{##1hyphenmins}{##2}%
4392     \fi}%
4393   \def\set@hyphenmins##1##2{%
4394     \lefthyphenmin##1\relax
4395     \righthyphenmin##2\relax}%
4396   \def\selectlanguage{%
4397     \errhelp{Selecting a language requires a package supporting it}%
4398     \errmessage{Not loaded}}%
4399   \let\foreignlanguage\selectlanguage
4400   \let\otherlanguage\selectlanguage
4401   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4402   \def\bbl@usehooks##1##2{% TODO. Temporary!!
4403     \def\setlocale{%
4404       \errhelp{Find an armchair, sit down and wait}%
4405       \errmessage{Not yet available}}%
4406     \let\uselocale\setlocale
4407     \let\locale\setlocale
4408     \let\selectlocale\setlocale
4409     \let\localename\setlocale
4410     \let\textlocale\setlocale
4411     \let\textlanguage\setlocale
4412     \let\languagetext\setlocale}
4413   \begingroup
4414     \def\AddBabelHook#1#2{%
4415       \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4416         \def\next{\toks1}%
4417       \else
4418         \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4419       \fi
4420     \next}
4421   \ifx\directlua\@undefined
4422     \ifx\XeTeXinputencoding\@undefined\else
4423       \input xebabel.def
4424     \fi
4425   \else
4426     \input luababel.def
4427   \fi
4428   \openin1 = babel-\bbl@format.cfg
4429   \ifeof1
4430   \else

```



```

4431 \input babel-\bbl@format.cfg\relax
4432 \fi
4433 \closein1
4434 \endgroup
4435 \bbl@hook@loadkernel{switch.def}

```

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

```

4436 \openin1 = language.dat

```

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

```

4437 \def\language{english}%
4438 \ifeof1
4439 \message{I couldn't find the file language.dat,\space
4440          I will try the file hyphen.tex}
4441 \input hyphen.tex\relax
4442 \chardef\l@english\z@
4443 \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.

```

4444 \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.

```

4445 \loop
4446 \endlinechar\m@ne
4447 \read1 to \bbl@line
4448 \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.

```

4449 \if T\ifeof1F\fi T\relax
4450 \ifx\bbl@line\@empty\else
4451 \edef\bbl@line{\bbl@line\space\space\space}%
4452 \expandafter\process@line\bbl@line\relax
4453 \fi
4454 \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.

```

4455 \begingroup
4456 \def\bbl@elt#1#2#3#4{%
4457 \global\language=#2\relax
4458 \gdef\language{#1}%
4459 \def\bbl@elt##1##2##3##4{}}%
4460 \bbl@languages
4461 \endgroup
4462 \fi
4463 \closein1

```

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

```

4464 \if/\the\toks@/\else
4465 \errhelp{language.dat loads no language, only synonyms}
4466 \errmessage{Orphan language synonym}
4467 \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.

```

4468 \let\bbl@line\@undefined
4469 \let\process@line\@undefined
4470 \let\process@synonym\@undefined
4471 \let\process@language\@undefined
4472 \let\bbl@get@enc\@undefined
4473 \let\bbl@hyph@enc\@undefined
4474 \let\bbl@tempa\@undefined
4475 \let\bbl@hook@loadkernel\@undefined
4476 \let\bbl@hook@everylanguage\@undefined
4477 \let\bbl@hook@loadpatterns\@undefined
4478 \let\bbl@hook@loadexceptions\@undefined
4479 \</patterns>

```

Here the code for iniTeX ends.

## 8 Font handling with fontspec

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

```

4480 <<{*More package options}>> ≡
4481 \chardef\bbl@bidimode\z@
4482 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4483 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4484 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4485 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4486 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4487 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4488 <</More package options>>

```

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

At the time of this writing, fontspec shows a warning about there are languages not available, which some people think refers to babel, even if there is nothing wrong. Here is hack to patch fontspec to avoid the misleading (and mostly unuseful) message.

```

4489 <<{*Font selection}>> ≡
4490 \bbl@trace{Font handling with fontspec}
4491 \ifx\ExplSyntaxOn\@undefined\else
4492   \def\bbl@fs@warn@nx#1#2{% \bbl@tempfs is the original macro
4493     \in@{,#1,}{,no-script,language-not-exist,}%
4494     \ifin@ \else\bbl@tempfs@nx{#1}{#2}\fi}
4495   \def\bbl@fs@warn@nxx#1#2#3{%
4496     \in@{,#1,}{,no-script,language-not-exist,}%
4497     \ifin@ \else\bbl@tempfs@nxx{#1}{#2}{#3}\fi}
4498   \def\bbl@loadfontspec{%
4499     \let\bbl@loadfontspec\relax
4500     \ifx\fontspec\@undefined
4501       \usepackage{fontspec}%
4502       \fi}%
4503 \fi
4504 \@onlypreamble\babelfont
4505 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4506   \bbl@foreach{#1}{%
4507     \expandafter\ifx\csname date##1\endcsname\relax
4508       \IfFileExists{babel-##1.tex}%
4509       {\babelprovide{##1}}%
4510     }%
4511   \fi}%
4512 \edef\bbl@tempa{#1}%
4513 \def\bbl@tempb{#2}% Used by \bbl@bblfont

```

```

4514 \bbl@loadfontspec
4515 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4516 \bbl@bblfont}
4517 \newcommand\bbl@bblfont[2][{}% 1=features 2=fontname, @font=rm|sf|tt
4518 \bbl@ifunset{\bbl@tempb family}%
4519 {\bbl@providfam{\bbl@tempb}}%
4520 {}%
4521 % For the default font, just in case:
4522 \bbl@ifunset{\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4523 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4524 {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
4525 \bbl@exp{%
4526 \let<\bbl@\bbl@tempb dflt@\languagename>\<\bbl@\bbl@tempb dflt@>%
4527 \\\bbl@font@set<\bbl@\bbl@tempb dflt@\languagename>%
4528 \<\bbl@tempb default>\<\bbl@tempb family>}}%
4529 {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *srt
4530 \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:

```

4531 \def\bbl@providfam#1{%
4532 \bbl@exp{%
4533 \\\newcommand<#1default>{}% Just define it
4534 \\\bbl@add@list\\bbl@font@fams{#1}%
4535 \\\DeclareRobustCommand<#1family>{%
4536 \\\not@math@alphabet<#1family>\relax
4537 % \\\prepare@family@series@update{#1}<#1default>% TODO. Fails
4538 \\\fontfamily<#1default>%
4539 \<ifx>\\UseHooks\\undefined\<else>\\UseHook{#1family}\<fi>%
4540 \\\selectfont}%
4541 \\\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.

```

4542 \def\bbl@nostdfont#1{%
4543 \bbl@ifunset{\bbl@WFF@\f@family}%
4544 {\bbl@csarg\gdef{WFF@\f@family}}{}% Flag, to avoid dupl warns
4545 \bbl@infowarn{The current font is not a babel standard family:\%
4546 #1%
4547 \fontname\font\\%
4548 There is nothing intrinsically wrong with this warning, and\\%
4549 you can ignore it altogether if you do not need these\\%
4550 families. But if they are used in the document, you should be\\%
4551 aware 'babel' will not set Script and Language for them, so\\%
4552 you may consider defining a new family with \string\babelfont.\\%
4553 See the manual for further details about \string\babelfont.\\%
4554 Reported}}
4555 {}}%
4556 \gdef\bbl@switchfont{%
4557 \bbl@ifunset{\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4558 \bbl@exp{% eg Arabic -> arabic
4559 \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}}%
4560 \bbl@foreach\bbl@font@fams{%
4561 \bbl@ifunset{\bbl@##1dflt@\languagename}% (1) language?
4562 {\bbl@ifunset{\bbl@##1dflt*~\bbl@tempa}% (2) from script?
4563 {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
4564 {}% 123=F - nothing!
4565 {\bbl@exp{% 3=T - from generic
4566 \global\let<\bbl@##1dflt@\languagename>%
4567 \<\bbl@##1dflt@>}}}%
4568 {\bbl@exp{% 2=T - from script
4569 \global\let<\bbl@##1dflt@\languagename>%
4570 \<\bbl@##1dflt*~\bbl@tempa>}}}%
4571 {}}% 1=T - language, already defined
4572 \def\bbl@tempa{\bbl@nostdfont{}}% TODO. Don't use \bbl@tempa

```

```

4573 \bbl@foreach\bbl@font@fams{%      don't gather with prev for
4574   \bbl@ifunset{\bbl@##1dflt@\languagename}%
4575   {\bbl@cs{famrst@##1}%
4576    \global\bbl@csarg\let{famrst@##1}\relax}%
4577   {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4578    \\bbl@add\\originalTeX{%
4579     \\bbl@font@rst{\bbl@cl{##1dflt}}}%
4580     \<##1default>\<##1family>{##1}}}%
4581    \\bbl@font@set{\bbl@##1dflt@\languagename}% the main part!
4582    \<##1default>\<##1family>}}}%
4583 \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.

```

4584 \ifx\family\undefined\else      % if latex
4585 \ifcase\bbl@engine                % if pdftex
4586   \let\bbl@ckeckstdfonts\relax
4587 \else
4588   \def\bbl@ckeckstdfonts{%
4589     \begingroup
4590     \global\let\bbl@ckeckstdfonts\relax
4591     \let\bbl@tempa\empty
4592     \bbl@foreach\bbl@font@fams{%
4593       \bbl@ifunset{\bbl@##1dflt@}%
4594       {\nameuse{##1family}}%
4595       \bbl@csarg\gdef{WFF@f@family}}}% Flag
4596       \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= \f@family\\}%
4597        \space\space\fontname\font\\}%
4598       \bbl@csarg\xdef{##1dflt@}{\f@family}%
4599       \expandafter\xdef\csname ##1default\endcsname{\f@family}}}%
4600       {}}%
4601   \ifx\bbl@tempa\empty\else
4602     \bbl@infowarn{The following font families will use the default\\%
4603       settings for all or some languages:\\%
4604       \bbl@tempa
4605       There is nothing intrinsically wrong with it, but\\%
4606       'babel' will no set Script and Language, which could\\%
4607       be relevant in some languages. If your document uses\\%
4608       these families, consider redefining them with \string\babelfont.\\%
4609       Reported}%
4610     \fi
4611   \endgroup}
4612 \fi
4613 \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.

```

4614 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4615   \bbl@xin@{<>}{#1}%
4616   \ifin@
4617     \bbl@exp{\\bbl@fontspec@set\\#1\expandafter@gobbletwo#1\\#3}%
4618   \fi
4619   \bbl@exp{%
4620     \def\\#2{#1}%          eg, \rmdefault{\bbl@rmdflt@lang}
4621     \\bbl@ifsamestring{#2}{\f@family}%
4622     {\\#3%
4623      \\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}}}%
4624     \let\\bbl@tempa\relax}%
4625   {}}%
4626 %   TODO - next should be global?, but even local does its job. I'm
4627 %   still not sure -- must investigate:
4628 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily

```

```

4629 \let\bbl@tempe\bbl@mapselect
4630 \let\bbl@mapselect\relax
4631 \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4632 \let#4\@empty % Make sure \renewfontfamily is valid
4633 \bbl@exp{%
4634 \let\\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
4635 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4636 {\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4637 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4638 {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4639 \let\\bbl@tempfs@nx<__fontspec_warning:nx>%
4640 \let<__fontspec_warning:nx>\\bbl@fs@warn@nx
4641 \let\\bbl@tempfs@nxx<__fontspec_warning:nxx>%
4642 \let<__fontspec_warning:nxx>\\bbl@fs@warn@nxx
4643 \\renewfontfamily\\#4%
4644 [\bbl@cl{lsys},#2]}{#3}% ie \bbl@exp{.}{#3}
4645 \bbl@exp{%
4646 \let<__fontspec_warning:nx>\\bbl@tempfs@nx
4647 \let<__fontspec_warning:nxx>\\bbl@tempfs@nxx}%
4648 \begingroup
4649 #4%
4650 \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4651 \endgroup
4652 \let#4\bbl@temp@fam
4653 \bbl@exp{\let<\bbl@stripslash#4\space>}\bbl@temp@pfam
4654 \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.

```

4655 \def\bbl@font@rst#1#2#3#4{%
4656 \bbl@csarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}

```

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

```

4657 \def\bbl@font@fams{rm,sf,tt}
4658 <</Font selection>>

```

## 9 Hooks for XeTeX and LuaTeX

### 9.1 XeTeX

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

```

4659 <<{*Footnote changes}>> ≡
4660 \bbl@trace{Bidi footnotes}
4661 \ifnum\bbl@bidimode>\z@
4662 \def\bbl@footnote#1#2#3{%
4663 \ifnextchar[%
4664 {\bbl@footnote@o{#1}{#2}{#3}}%
4665 {\bbl@footnote@x{#1}{#2}{#3}}%
4666 \long\def\bbl@footnote@x#1#2#3#4{%
4667 \bgroup
4668 \select@language@x{\bbl@main@language}%
4669 \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4670 \egroup}
4671 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4672 \bgroup
4673 \select@language@x{\bbl@main@language}%
4674 \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4675 \egroup}
4676 \def\bbl@footnotetext#1#2#3{%
4677 \ifnextchar[%
4678 {\bbl@footnotetext@o{#1}{#2}{#3}}%

```

```

4679     {\bbl@footnotetext@x{#1}{#2}{#3}}
4680 \long\def\bbl@footnotetext@x#1#2#3#4{%
4681   \bgroup
4682   \select@language@x{\bbl@main@language}%
4683   \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4684   \egroup}
4685 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4686   \bgroup
4687   \select@language@x{\bbl@main@language}%
4688   \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4689   \egroup}
4690 \def\BabelFootnote#1#2#3#4{%
4691   \ifx\bbl@fn@footnote\@undefined
4692     \let\bbl@fn@footnote\footnote
4693   \fi
4694   \ifx\bbl@fn@footnotetext\@undefined
4695     \let\bbl@fn@footnotetext\footnotetext
4696   \fi
4697   \bbl@ifblank{#2}%
4698   {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4699    \@namedef{\bbl@stripslash#1text}%
4700    {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4701   {\def#1{\bbl@exp{\bbl@footnote{\bbl@foreignlanguage{#2}}}{#3}{#4}}%
4702    \@namedef{\bbl@stripslash#1text}%
4703    {\bbl@exp{\bbl@footnotetext{\bbl@foreignlanguage{#2}}}{#3}{#4}}}%
4704 \fi
4705 <</Footnote changes>>

```

Now, the code.

```

4706 <*\xetex>
4707 \def\BabelStringsDefault{unicode}
4708 \let\xebbl@stop\relax
4709 \AddBabelHook{xetex}{encodedcommands}{%
4710   \def\bbl@tempa{#1}%
4711   \ifx\bbl@tempa\@empty
4712     \XeTeXinputencoding"bytes"%
4713   \else
4714     \XeTeXinputencoding"#1"%
4715   \fi
4716   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4717 \AddBabelHook{xetex}{stopcommands}{%
4718   \xebbl@stop
4719   \let\xebbl@stop\relax}
4720 \def\bbl@intraspace#1 #2 #3@@{%
4721   \bbl@csarg\gdef{xeisp@\languagename}%
4722   {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4723 \def\bbl@intrapenalty#1@@{%
4724   \bbl@csarg\gdef{xeipn@\languagename}%
4725   {\XeTeXlinebreakpenalty #1\relax}}
4726 \def\bbl@provide@intraspace{%
4727   \bbl@xin@{/s}{/\bbl@cl{lnbrk}}}%
4728   \ifin@ \else \bbl@xin@{/c}{/\bbl@cl{lnbrk}} \fi
4729   \ifin@
4730     \bbl@ifunset{\bbl@intsp@\languagename}{}%
4731     {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
4732       \ifx\bbl@KVP@intraspace\@nnil
4733         \bbl@exp{%
4734           \bbl@intraspace\bbl@cl{intsp}\@@@}%
4735       \fi
4736       \ifx\bbl@KVP@intrapenalty\@nnil
4737         \bbl@intrapenalty0@@@
4738       \fi
4739     \fi

```

```

4740 \ifx\bb1@KVP@intraspace\@nnil\else % We may override the ini
4741 \expandafter\bb1@intraspace\bb1@KVP@intraspace\@@
4742 \fi
4743 \ifx\bb1@KVP@intrapenalty\@nnil\else
4744 \expandafter\bb1@intrapenalty\bb1@KVP@intrapenalty\@@
4745 \fi
4746 \bb1@exp{%
4747 % TODO. Execute only once (but redundant):
4748 \\bb1@add\<extras\language>{%
4749 \XeTeXlinebreaklocale "\bb1@cl{tbc}"%
4750 \<bb1@xeisp@\language>%
4751 \<bb1@xeipn@\language>}%
4752 \\bb1@tglobal\<extras\language>%
4753 \\bb1@add\<noextras\language>{%
4754 \XeTeXlinebreaklocale ""}%
4755 \\bb1@tglobal\<noextras\language>}%
4756 \ifx\bb1@ispace\@undefined
4757 \gdef\bb1@ispace{\bb1@cl{xeisp}}%
4758 \ifx\AtBeginDocument\@notprerr
4759 \expandafter\@secondoftwo % to execute right now
4760 \fi
4761 \AtBeginDocument{\bb1@patchfont{\bb1@ispace}}%
4762 \fi}%
4763 \fi}
4764 \ifx\DisableBabelHook\@undefined\endinput\fi
4765 \AddBabelHook{babel-fontspec}{afterextras}{\bb1@switchfont}
4766 \AddBabelHook{babel-fontspec}{beforestart}{\bb1@ckeckstdfonts}
4767 \DisableBabelHook{babel-fontspec}
4768 <<Font selection>>
4769 \def\bb1@provide@extra#1{}
4770 </xetex>

```

## 9.2 Layout

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

\bb1@startskip and \bb1@endskip are available to package authors. Thanks to the T<sub>E</sub>X expansion mechanism the following constructs are valid: \adim\bb1@startskip, \advance\bb1@startskip\adim, \bb1@startskip\adim.

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

```

4771 (*xetex | texxet)
4772 \providecommand\bb1@provide@intraspace{}
4773 \bb1@trace{Redefinitions for bidi layout}
4774 \def\bb1@sspre@caption{%
4775 \bb1@exp{\everyhbox{\bb1@texdir\bb1@cs{wdir@bb1@main@language}}}}
4776 \ifx\bb1@opt@layout\@nnil\else % if layout=..
4777 \def\bb1@startskip{\ifcase\bb1@thepardir\leftskip\else\rightskip\fi}
4778 \def\bb1@endskip{\ifcase\bb1@thepardir\rightskip\else\leftskip\fi}
4779 \ifx\bb1@beforeforeign\leavevmode % A poor test for bidi=
4780 \def\@hangfrom#1{%
4781 \setbox\@tempboxa\hbox{#1}}%
4782 \hangindent\ifcase\bb1@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4783 \noindent\box\@tempboxa}
4784 \def\raggedright{%
4785 \let\@centercr
4786 \bb1@startskip\z@skip
4787 \@rightskip\@flushglue
4788 \bb1@endskip\@rightskip
4789 \parindent\z@
4790 \parfillskip\bb1@startskip}
4791 \def\raggedleft{%
4792 \let\@centercr
4793 \bb1@startskip\@flushglue

```

```

4794 \bbl@endskip\z@skip
4795 \parindent\z@
4796 \parfillskip\bbl@endskip}
4797 \fi
4798 \IfBabelLayout{lists}
4799 {\bbl@sreplace\list
4800 {\totalleftmargin\leftmargin}{\totalleftmargin\bbl@listleftmargin}%
4801 \def\bbl@listleftmargin{%
4802 \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4803 \ifcase\bbl@engine
4804 \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4805 \def\p@enumii{\p@enumii}\theenumii{}\%
4806 \fi
4807 \bbl@sreplace\@verbatim
4808 {\leftskip\totalleftmargin}%
4809 {\bbl@startskip\textwidth
4810 \advance\bbl@startskip-\linewidth}%
4811 \bbl@sreplace\@verbatim
4812 {\rightskip\z@skip}%
4813 {\bbl@endskip\z@skip}}%
4814 {}
4815 \IfBabelLayout{contents}
4816 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4817 \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4818 {}
4819 \IfBabelLayout{columns}
4820 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4821 \def\bbl@outputbox#1{%
4822 \hb@xt@\textwidth{%
4823 \hskip\columnwidth
4824 \hfil
4825 {\normalcolor\vrule \@width\columnseprule}%
4826 \hfil
4827 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4828 \hskip-\textwidth
4829 \hb@xt@\columnwidth{\box\@outputbox \hss}%
4830 \hskip\columnsep
4831 \hskip\columnwidth}}}%
4832 {}
4833 <<Footnote changes>>
4834 \IfBabelLayout{footnotes}%
4835 {\BabelFootnote\footnote\languagename{}\}%
4836 \BabelFootnote\localfootnote\languagename{}\}%
4837 \BabelFootnote\mainfootnote{}\}%
4838 {}

```

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.

```

4839 \IfBabelLayout{counters*}%
4840 {\bbl@add\bbl@opt@layout{.counters.}%
4841 \AddToHook{shipout/before}{%
4842 \let\bbl@tempa\babelsublr
4843 \let\babelsublr\@firstofone
4844 \let\bbl@save@thepage\thepage
4845 \protected@edef\thepage{\thepage}%
4846 \let\babelsublr\bbl@tempa}%
4847 \AddToHook{shipout/after}{%
4848 \let\thepage\bbl@save@thepage}}{}
4849 \IfBabelLayout{counters}%
4850 {\let\bbl@latinarabic=\@arabic
4851 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4852 \let\bbl@asciroman=\@roman
4853 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%

```



```

4854 \let\bbl@asciiRoman=\@Roman
4855 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}{}}
4856 \fi % end if layout
4857 \xetex|texet)

```

### 9.3 8-bit TeX

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

```

4858 (*texxet)
4859 \def\bbl@provide@extra#1{%
4860 % == auto-select encoding ==
4861 \ifx\bbl@encoding@select@off\@empty\else
4862 \bbl@ifunset{\bbl@encoding@#1}%
4863 {\def\@elt##1{,##1,}%
4864 \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
4865 \count@\z@
4866 \bbl@foreach\bbl@tempe{%
4867 \def\bbl@tempd{##1}% Save last declared
4868 \advance\count@\@ne}%
4869 \ifnum\count@>\@ne
4870 \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
4871 \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
4872 \bbl@replace\bbl@tempa{ },}%
4873 \global\bbl@csarg\let{encoding@#1}\@empty
4874 \bbl@xin@{,\bbl@tempd,}{,\bbl@tempa,}%
4875 \ifin\@else % if main encoding included in ini, do nothing
4876 \let\bbl@tempb\relax
4877 \bbl@foreach\bbl@tempa{%
4878 \ifx\bbl@tempb\relax
4879 \bbl@xin@{,##1,}{,\bbl@tempe,}%
4880 \ifin\def\bbl@tempb{##1}\fi
4881 \fi}%
4882 \ifx\bbl@tempb\relax\else
4883 \bbl@exp{%
4884 \global\<\bbl@add>\<\bbl@preextras@#1>\<\bbl@encoding@#1>}%
4885 \gdef\<\bbl@encoding@#1>{%
4886 \\\babel@save\\f@encoding
4887 \\\bbl@add\\originalTeX{\\selectfont}%
4888 \\\fontencoding{\bbl@tempb}%
4889 \\\selectfont}}%
4890 \fi
4891 \fi
4892 \fi}%
4893 {}%
4894 \fi}
4895 \xetex)

```

### 9.4 LuaTeX

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

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

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

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

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

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

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

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

```

4896 <*luatex>
4897 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4898 \bbl@trace{Read language.dat}
4899 \ifx\bbl@readstream\undefined
4900   \csname newread\endcsname\bbl@readstream
4901 \fi
4902 \begingroup
4903   \toks@{}
4904   \count@ \z@ % 0=start, 1=0th, 2=normal
4905   \def\bbl@process@line#1#2 #3 #4 {%
4906     \ifx=#1%
4907       \bbl@process@synonym{#2}%
4908     \else
4909       \bbl@process@language{#1#2}{#3}{#4}%
4910     \fi
4911     \ignorespaces}
4912 \def\bbl@manylang{%
4913   \ifnum\bbl@last>\@ne
4914     \bbl@info{Non-standard hyphenation setup}%
4915   \fi
4916   \let\bbl@manylang\relax}
4917 \def\bbl@process@language#1#2#3{%
4918   \ifcase\count@
4919     \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4920   \or
4921     \count@\tw@
4922   \fi
4923   \ifnum\count@=\tw@
4924     \expandafter\addlanguage\csname l@#1\endcsname
4925     \language\allocationnumber
4926     \chardef\bbl@last\allocationnumber
4927     \bbl@manylang
4928     \let\bbl@elt\relax
4929     \xdef\bbl@languages{%
4930       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4931   \fi
4932   \the\toks@
4933   \toks@{}}
4934 \def\bbl@process@synonym@aux#1#2{%
4935   \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4936   \let\bbl@elt\relax
4937   \xdef\bbl@languages{%
4938     \bbl@languages\bbl@elt{#1}{#2}{}}}%
4939 \def\bbl@process@synonym#1{%
4940   \ifcase\count@

```

```

4941 \toks@expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4942 \or
4943 \ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}}%
4944 \else
4945 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4946 \fi}
4947 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4948 \chardef\l@english\z@
4949 \chardef\l@USenglish\z@
4950 \chardef\bbl@last\z@
4951 \global\@namedef{bbl@hyphendata@0}{\hyphen.tex}{}
4952 \gdef\bbl@languages{%
4953 \bbl@elt{english}{0}{\hyphen.tex}{}%
4954 \bbl@elt{USenglish}{0}{}}%
4955 \else
4956 \global\let\bbl@languages@format\bbl@languages
4957 \def\bbl@elt#1#2#3#4{% Remove all except language 0
4958 \ifnum#2>\z@\else
4959 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4960 \fi}%
4961 \xdef\bbl@languages{\bbl@languages}%
4962 \fi
4963 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}} % Define flags
4964 \bbl@languages
4965 \openin\bbl@readstream=language.dat
4966 \ifEOF\bbl@readstream
4967 \bbl@warning{I couldn't find language.dat. No additional\\%
4968 patterns loaded. Reported}%
4969 \else
4970 \loop
4971 \endlinechar\m@ne
4972 \read\bbl@readstream to \bbl@line
4973 \endlinechar`^^M
4974 \if T\ifEOF\bbl@readstream F\fi T\relax
4975 \ifx\bbl@line\@empty\else
4976 \edef\bbl@line{\bbl@line\space\space\space}%
4977 \expandafter\bbl@process@line\bbl@line\relax
4978 \fi
4979 \repeat
4980 \fi
4981 \closein\bbl@readstream
4982 \endgroup
4983 \bbl@trace{Macros for reading patterns files}
4984 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
4985 \ifx\babelcatcodetablenum\@undefined
4986 \ifx\newcatcodetable\@undefined
4987 \def\babelcatcodetablenum{5211}
4988 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4989 \else
4990 \newcatcodetable\babelcatcodetablenum
4991 \newcatcodetable\bbl@pattcodes
4992 \fi
4993 \else
4994 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4995 \fi
4996 \def\bbl@luapatterns#1#2{%
4997 \bbl@get@enc#1::\@@
4998 \setbox\z@\hbox\bgroup
4999 \begingroup
5000 \savecatcodetable\babelcatcodetablenum\relax
5001 \initcatcodetable\bbl@pattcodes\relax
5002 \catcodetable\bbl@pattcodes\relax
5003 \catcode\#=6 \catcode\$=3 \catcode\&=4 \catcode\^=7

```

```

5004 \catcode`\_ =8 \catcode`\{ =1 \catcode`\} =2 \catcode`\~ =13
5005 \catcode`\@ =11 \catcode`\^^I =10 \catcode`\^^J =12
5006 \catcode`\< =12 \catcode`\> =12 \catcode`\* =12 \catcode`\.=12
5007 \catcode`\- =12 \catcode`\ / =12 \catcode`\[ =12 \catcode`\] =12
5008 \catcode`\' =12 \catcode`\' =12 \catcode`\ " =12
5009 \input #1\relax
5010 \catcodetable\babelcatcodetablenum\relax
5011 \endgroup
5012 \def\bbl@tempa{#2}%
5013 \ifx\bbl@tempa\empty\else
5014 \input #2\relax
5015 \fi
5016 \egroup}%
5017 \def\bbl@patterns@lua#1{%
5018 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5019 \csname l@#1\endcsname
5020 \edef\bbl@tempa{#1}%
5021 \else
5022 \csname l@#1:\f@encoding\endcsname
5023 \edef\bbl@tempa{#1:\f@encoding}%
5024 \fi\relax
5025 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5026 \@ifundefined{bbl@hyphendata@the\language}%
5027 { \def\bbl@elt##1##2##3##4{%
5028 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
5029 \def\bbl@tempb{##3}%
5030 \ifx\bbl@tempb\empty\else % if not a synonymous
5031 \def\bbl@tempc{{##3}{##4}}%
5032 \fi
5033 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5034 \fi}%
5035 \bbl@languages
5036 \@ifundefined{bbl@hyphendata@the\language}%
5037 {\bbl@info{No hyphenation patterns were set for\%
5038 language '\bbl@tempa'. Reported}}%
5039 {\expandafter\expandafter\expandafter\bbl@luapatterns
5040 \csname bbl@hyphendata@the\language\endcsname}}}%
5041 \endinput\fi
5042 % Here ends \ifx\AddBabelHook\@undefined
5043 % A few lines are only read by hyphen.cfg
5044 \ifx\DisableBabelHook\@undefined
5045 \AddBabelHook{luatex}{everylanguage}{%
5046 \def\process@language##1##2##3{%
5047 \def\process@line####1####2 ####3 ####4 {}}%
5048 \AddBabelHook{luatex}{loadpatterns}{%
5049 \input #1\relax
5050 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5051 {{#1}}}%
5052 \AddBabelHook{luatex}{loadexceptions}{%
5053 \input #1\relax
5054 \def\bbl@tempb##1##2{{##1}{##2}}%
5055 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5056 {\expandafter\expandafter\expandafter\bbl@tempb
5057 \csname bbl@hyphendata@the\language\endcsname}}%
5058 \endinput\fi
5059 % Here stops reading code for hyphen.cfg
5060 % The following is read the 2nd time it's loaded
5061 \begingroup % TODO - to a lua file
5062 \catcode`\% =12
5063 \catcode`\' =12
5064 \catcode`\ " =12
5065 \catcode`\: =12
5066 \directlua{

```

```

5067 Babel = Babel or {}
5068 function Babel.bytes(line)
5069     return line:gsub("(.)",
5070         function (chr) return unicode.utf8.char(string.byte(chr)) end)
5071 end
5072 function Babel.begin_process_input()
5073     if luatexbase and luatexbase.add_to_callback then
5074         luatexbase.add_to_callback('process_input_buffer',
5075             Babel.bytes, 'Babel.bytes')
5076     else
5077         Babel.callback = callback.find('process_input_buffer')
5078         callback.register('process_input_buffer', Babel.bytes)
5079     end
5080 end
5081 function Babel.end_process_input ()
5082     if luatexbase and luatexbase.remove_from_callback then
5083         luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5084     else
5085         callback.register('process_input_buffer', Babel.callback)
5086     end
5087 end
5088 function Babel.addpatterns(pp, lg)
5089     local lg = lang.new(lg)
5090     local pats = lang.patterns(lg) or ''
5091     lang.clear_patterns(lg)
5092     for p in pp:gmatch('[^%s]+') do
5093         ss = ''
5094         for i in string.utfcharacters(p:gsub('%d', '')) do
5095             ss = ss .. '%d?' .. i
5096         end
5097         ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
5098         ss = ss:gsub('%.%%d%?$', '%%.')
5099         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5100         if n == 0 then
5101             tex.sprint(
5102                 [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5103                 .. p .. [[{ }]])
5104             pats = pats .. ' ' .. p
5105         else
5106             tex.sprint(
5107                 [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5108                 .. p .. [[{ }]])
5109         end
5110     end
5111     lang.patterns(lg, pats)
5112 end
5113 Babel.characters = Babel.characters or {}
5114 Babel.ranges = Babel.ranges or {}
5115 function Babel.hlist_has_bidi(head)
5116     local has_bidi = false
5117     local ranges = Babel.ranges
5118     for item in node.traverse(head) do
5119         if item.id == node.id'glyph' then
5120             local itemchar = item.char
5121             local chardata = Babel.characters[itemchar]
5122             local dir = chardata and chardata.d or nil
5123             if not dir then
5124                 for nn, et in ipairs(ranges) do
5125                     if itemchar < et[1] then
5126                         break
5127                     elseif itemchar <= et[2] then
5128                         dir = et[3]
5129                         break

```

```

5130         end
5131     end
5132 end
5133 if dir and (dir == 'al' or dir == 'r') then
5134     has_bidi = true
5135 end
5136 end
5137 end
5138 return has_bidi
5139 end
5140 function Babel.set_chranges_b (script, chrng)
5141     if chrng == '' then return end
5142     texio.write('Replacing ' .. script .. ' script ranges')
5143     Babel.script_blocks[script] = {}
5144     for s, e in string.gmatch(chrng..' ', '(.-%.-%.-%s)') do
5145         table.insert(
5146             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5147     end
5148 end
5149 function Babel.discard_sublr(str)
5150     if str:find( [[\string\indexentry]] ) and
5151        str:find( [[\string\babelsublr]] ) then
5152         str = str:gsub( [[\string\babelsublr%s*(%b{})]],
5153             function(m) return m:sub(2,-2) end )
5154     end
5155     return str
5156 end
5157 }
5158 \endgroup
5159 \ifx\newattribute\undefined\else
5160     \newattribute\bbl@attr@locale
5161     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5162     \AddBabelHook{luatex}{beforeextras}{%
5163         \setattribute\bbl@attr@locale\localeid}
5164 \fi
5165 \def\BabelStringsDefault{unicode}
5166 \let\luabbl@stop\relax
5167 \AddBabelHook{luatex}{encodedcommands}{%
5168     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5169     \ifx\bbl@tempa\bbl@tempb\else
5170         \directlua{Babel.begin_process_input()}%
5171         \def\luabbl@stop{%
5172             \directlua{Babel.end_process_input()}}%
5173     \fi}%
5174 \AddBabelHook{luatex}{stopcommands}{%
5175     \luabbl@stop
5176     \let\luabbl@stop\relax}
5177 \AddBabelHook{luatex}{patterns}{%
5178     \ifundefined{bbl@hyphendata@the\language}%
5179         {\def\bbl@elt##1##2##3##4{%
5180             \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
5181             \def\bbl@tempb{##3}%
5182             \ifx\bbl@tempb@empty\else % if not a synonymous
5183                 \def\bbl@tempc{{##3}{##4}}}%
5184             \fi
5185             \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5186             \fi}%
5187     \bbl@languages
5188     \ifundefined{bbl@hyphendata@the\language}%
5189         {\bbl@info{No hyphenation patterns were set for\\%
5190             language '#2'. Reported}}%
5191     {\expandafter\expandafter\expandafter\bbl@luapatterns
5192         \csname bbl@hyphendata@the\language\endcsname}}}%

```

```

5193 \@ifundefined{bbl@patterns@}{}%
5194 \begingroup
5195 \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5196 \ifin@else
5197 \ifx\bbl@patterns@\@empty\else
5198 \directlua{ Babel.addpatterns(
5199 [[\bbl@patterns@]], \number\language) }%
5200 \fi
5201 \@ifundefined{bbl@patterns@#1}%
5202 \@empty
5203 {\directlua{ Babel.addpatterns(
5204 [[\space\csname bbl@patterns@#1\endcsname]],
5205 \number\language) }}%
5206 \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5207 \fi
5208 \endgroup}%
5209 \bbl@exp{%
5210 \bbl@ifunset{bbl@prehc@\languagename}{}%
5211 {\bbl@ifblank{\bbl@cs{prehc@\languagename}}{}}%
5212 {\prehyphenchar=\bbl@c1{prehc}\relax}}}%

```

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

```

5213 \@onlypreamble\babelpatterns
5214 \AtEndOfPackage{%
5215 \newcommand\babelpatterns[2][\@empty]{%
5216 \ifx\bbl@patterns@\relax
5217 \let\bbl@patterns@\@empty
5218 \fi
5219 \ifx\bbl@pttnlist@\@empty\else
5220 \bbl@warning{%
5221 You must not intermingle \string\selectlanguage\space and\\%
5222 \string\babelpatterns\space or some patterns will not\\%
5223 be taken into account. Reported}%
5224 \fi
5225 \ifx\@empty#1%
5226 \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5227 \else
5228 \edef\bbl@tempb{\zap@space#1 \@empty}%
5229 \bbl@for\bbl@tempa\bbl@tempb{%
5230 \bbl@fixname\bbl@tempa
5231 \bbl@iflanguage\bbl@tempa{%
5232 \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5233 \@ifundefined{bbl@patterns@\bbl@tempa}%
5234 \@empty
5235 {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5236 #2}}}%
5237 \fi}}

```

## 9.5 Southeast Asian scripts

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

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

```

5238 % TODO - to a lua file
5239 \directlua{
5240 Babel = Babel or {}
5241 Babel.linebreaking = Babel.linebreaking or {}
5242 Babel.linebreaking.before = {}
5243 Babel.linebreaking.after = {}

```

```

5244 Babel.locale = {} % Free to use, indexed by \localeid
5245 function Babel.linebreaking.add_before(func, pos)
5246     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5247     if pos == nil then
5248         table.insert(Babel.linebreaking.before, func)
5249     else
5250         table.insert(Babel.linebreaking.before, pos, func)
5251     end
5252 end
5253 function Babel.linebreaking.add_after(func)
5254     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5255     table.insert(Babel.linebreaking.after, func)
5256 end
5257 }
5258 \def\bbl@intraspace#1 #2 #3\@@{%
5259     \directlua{
5260         Babel = Babel or {}
5261         Babel.intraspaces = Babel.intraspaces or {}
5262         Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5263             {b = #1, p = #2, m = #3}
5264         Babel.locale_props[\the\localeid].intraspace = %
5265             {b = #1, p = #2, m = #3}
5266     }}
5267 \def\bbl@intrapenalty#1\@@{%
5268     \directlua{
5269         Babel = Babel or {}
5270         Babel.intrapenalties = Babel.intrapenalties or {}
5271         Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5272         Babel.locale_props[\the\localeid].intrapenalty = #1
5273     }}
5274 \begingroup
5275 \catcode`\%=12
5276 \catcode`\^=14
5277 \catcode`\'=12
5278 \catcode`\~=12
5279 \gdef\bbl@seaintraspace{^
5280     \let\bbl@seaintraspace\relax
5281     \directlua{
5282         Babel = Babel or {}
5283         Babel.sea_enabled = true
5284         Babel.sea_ranges = Babel.sea_ranges or {}
5285         function Babel.set_chrngs (script, chrng)
5286             local c = 0
5287             for s, e in string.gmatch(chrng..' ', '(.-%.(-)%s') do
5288                 Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5289                 c = c + 1
5290             end
5291         end
5292         function Babel.sea_disc_to_space (head)
5293             local sea_ranges = Babel.sea_ranges
5294             local last_char = nil
5295             local quad = 655360 ^% 10 pt = 655360 = 10 * 65536
5296             for item in node.traverse(head) do
5297                 local i = item.id
5298                 if i == node.id'glyph' then
5299                     last_char = item
5300                 elseif i == 7 and item.subtype == 3 and last_char
5301                     and last_char.char > 0x0C99 then
5302                     quad = font.getfont(last_char.font).size
5303                     for lg, rg in pairs(sea_ranges) do
5304                         if last_char.char > rg[1] and last_char.char < rg[2] then
5305                             lg = lg:sub(1, 4) ^% Remove trailing number of, eg, Cyril1
5306                             local intraspace = Babel.intraspaces[lg]

```



```

5307         local intrapenalty = Babel.intrapenalties[lg]
5308         local n
5309         if intrapenalty ~= 0 then
5310             n = node.new(14, 0)    ^% penalty
5311             n.penalty = intrapenalty
5312             node.insert_before(head, item, n)
5313         end
5314         n = node.new(12, 13)    ^% (glue, spaceskip)
5315         node.setglue(n, intraspace.b * quad,
5316                        intraspace.p * quad,
5317                        intraspace.m * quad)
5318         node.insert_before(head, item, n)
5319         node.remove(head, item)
5320     end
5321 end
5322 end
5323 end
5324 end
5325 }^^
5326 \bbl@luahyphenate}

```

## 9.6 CJK line breaking

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

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

```

5327 \catcode`\%=14
5328 \gdef\bbl@cjkintraspacespace{%
5329   \let\bbl@cjkintraspacespace\relax
5330   \directlua{
5331     Babel = Babel or {}
5332     require('babel-data-cjk.lua')
5333     Babel.cjk_enabled = true
5334     function Babel.cjk_linebreak(head)
5335       local GLYPH = node.id'glyph'
5336       local last_char = nil
5337       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5338       local last_class = nil
5339       local last_lang = nil
5340
5341       for item in node.traverse(head) do
5342         if item.id == GLYPH then
5343
5344           local lang = item.lang
5345
5346           local LOCALE = node.get_attribute(item,
5347                                             Babel.attr_locale)
5348           local props = Babel.locale_props[LOCALE]
5349
5350           local class = Babel.cjk_class[item.char].c
5351
5352           if props.cjk_quotes and props.cjk_quotes[item.char] then
5353             class = props.cjk_quotes[item.char]
5354           end
5355
5356           if class == 'cp' then class = 'cl' end % ]] as CL
5357           if class == 'id' then class = 'I' end
5358
5359           local br = 0
5360           if class and last_class and Babel.cjk_breaks[last_class][class] then

```

```

5361         br = Babel.cjk_breaks[last_class][class]
5362     end
5363
5364     if br == 1 and props.linebreak == 'c' and
5365         lang ~= \the\l@nohyphenation\space and
5366         last_lang ~= \the\l@nohyphenation then
5367         local intrapenalty = props.intrapenalty
5368         if intrapenalty ~= 0 then
5369             local n = node.new(14, 0)    % penalty
5370             n.penalty = intrapenalty
5371             node.insert_before(head, item, n)
5372         end
5373         local intraspace = props.intraspace
5374         local n = node.new(12, 13)    % (glue, spaceskip)
5375         node.setglue(n, intraspace.b * quad,
5376                     intraspace.p * quad,
5377                     intraspace.m * quad)
5378         node.insert_before(head, item, n)
5379     end
5380
5381     if font.getfont(item.font) then
5382         quad = font.getfont(item.font).size
5383     end
5384     last_class = class
5385     last_lang = lang
5386     else % if penalty, glue or anything else
5387         last_class = nil
5388     end
5389 end
5390 lang.hyphenate(head)
5391 end
5392 }%
5393 \bbl@luahyphenate}
5394 \gdef\bbl@luahyphenate{%
5395 \let\bbl@luahyphenate\relax
5396 \directlua{
5397     luatexbase.add_to_callback('hyphenate',
5398     function (head, tail)
5399         if Babel.linebreaking.before then
5400             for k, func in ipairs(Babel.linebreaking.before) do
5401                 func(head)
5402             end
5403         end
5404         if Babel.cjk_enabled then
5405             Babel.cjk_linebreak(head)
5406         end
5407         lang.hyphenate(head)
5408         if Babel.linebreaking.after then
5409             for k, func in ipairs(Babel.linebreaking.after) do
5410                 func(head)
5411             end
5412         end
5413         if Babel.sea_enabled then
5414             Babel.sea_disc_to_space(head)
5415         end
5416     end,
5417     'Babel.hyphenate')
5418 }
5419 }
5420 \endgroup
5421 \def\bbl@provide@intraspace{%
5422     \bbl@ifunset{\bbl@intsp@language\language\language}%
5423     {\xexpandafter\ifx\csname bbl@intsp@language\endcsname\empty\else

```

```

5424 \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
5425 \ifin@ % cjk
5426 \bbl@cjk intraspace
5427 \directlua{
5428     Babel = Babel or {}
5429     Babel.locale_props = Babel.locale_props or {}
5430     Babel.locale_props[\the\localeid].linebreak = 'c'
5431 }%
5432 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@@}%
5433 \ifx\bbl@KVP@intrapenalty\@nnil
5434 \bbl@intrapenalty0\@@
5435 \fi
5436 \else % sea
5437 \bbl@seaintraspace
5438 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@@}%
5439 \directlua{
5440     Babel = Babel or {}
5441     Babel.sea_ranges = Babel.sea_ranges or {}
5442     Babel.set_chranges('\bbl@cl{sbcpr}',
5443         '\bbl@cl{chrng}')
5444 }%
5445 \ifx\bbl@KVP@intrapenalty\@nnil
5446 \bbl@intrapenalty0\@@
5447 \fi
5448 \fi
5449 \fi
5450 \ifx\bbl@KVP@intrapenalty\@nnil\else
5451 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5452 \fi}}

```

## 9.7 Arabic justification

```

5453 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5454 \def\bblar@chars{%
5455     0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5456     0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5457     0640,0641,0642,0643,0644,0645,0646,0647,0649}
5458 \def\bblar@elongated{%
5459     0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5460     063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5461     0649,064A}
5462 \begingroup
5463 \catcode`_ =11 \catcode`:=11
5464 \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5465 \endgroup
5466 \gdef\bbl@arabicjust{%
5467     \let\bbl@arabicjust\relax
5468     \newattribute\bblar@kashida
5469     \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5470     \bblar@kashida=\z@
5471     \bbl@patchfont{\bbl@parsejalt}}%
5472     \directlua{
5473         Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5474         Babel.arabic.elong_map[\the\localeid] = {}
5475         luatexbase.add_to_callback('post_linebreak_filter',
5476             Babel.arabic.justify, 'Babel.arabic.justify')
5477         luatexbase.add_to_callback('hpack_filter',
5478             Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5479     }%
5480 % Save both node lists to make replacement. TODO. Save also widths to
5481 % make computations
5482 \def\bblar@fetchjalt#1#2#3#4{%
5483     \bbl@exp{\bbl@foreach{#1}}{%

```

```

5484 \bbl@ifunset{bblar@JE@##1}%
5485 {\setbox\z@\hbox{^^^^200d\char"##1#2}}%
5486 {\setbox\z@\hbox{^^^^200d\char"@nameuse{bblar@JE@##1}#2}}%
5487 \directlua{%
5488   local last = nil
5489   for item in node.traverse(tex.box[0].head) do
5490     if item.id == node.id'glyph' and item.char > 0x600 and
5491       not (item.char == 0x200D) then
5492       last = item
5493     end
5494   end
5495   Babel.arabic.#3['##1#4'] = last.char
5496 }}}}
5497 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5498 % perhaps other tables (falt?, csw?). What about kaf? And diacritic
5499 % positioning?
5500 \gdef\bbl@parsejalt{%
5501   \ifx\addfontfeature\@undefined\else
5502     \bbl@xin@{/e}{/\bbl@c1{lnbrk}}%
5503     \ifin@
5504       \directlua{%
5505         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5506           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5507           tex.print([\string\csname\space bbl@parsejalti\endcsname])
5508         end
5509       }%
5510     \fi
5511   \fi}
5512 \gdef\bbl@parsejalti{%
5513   \begingroup
5514     \let\bbl@parsejalt\relax % To avoid infinite loop
5515     \edef\bbl@tempb{\fontid\font}%
5516     \bblar@nofswarn
5517     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5518     \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5519     \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5520     \addfontfeature{RawFeature+=jalt}%
5521     % \namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5522     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5523     \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5524     \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5525     \directlua{%
5526       for k, v in pairs(Babel.arabic.from) do
5527         if Babel.arabic.dest[k] and
5528           not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5529           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5530             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5531         end
5532       end
5533     }%
5534   \endgroup}
5535 %
5536 \begingroup
5537 \catcode`#=11
5538 \catcode`~ =11
5539 \directlua{
5540
5541 Babel.arabic = Babel.arabic or {}
5542 Babel.arabic.from = {}
5543 Babel.arabic.dest = {}
5544 Babel.arabic.justify_factor = 0.95
5545 Babel.arabic.justify_enabled = true
5546

```

```

5547 function Babel.arabic.justify(head)
5548   if not Babel.arabic.justify_enabled then return head end
5549   for line in node.traverse_id(node.id'hlist', head) do
5550     Babel.arabic.justify_hlist(head, line)
5551   end
5552   return head
5553 end
5554
5555 function Babel.arabic.justify_hbox(head, gc, size, pack)
5556   local has_inf = false
5557   if Babel.arabic.justify_enabled and pack == 'exactly' then
5558     for n in node.traverse_id(12, head) do
5559       if n.stretch_order > 0 then has_inf = true end
5560     end
5561     if not has_inf then
5562       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5563     end
5564   end
5565   return head
5566 end
5567
5568 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5569   local d, new
5570   local k_list, k_item, pos_inline
5571   local width, width_new, full, k_curr, wt_pos, goal, shift
5572   local subst_done = false
5573   local elong_map = Babel.arabic.elong_map
5574   local last_line
5575   local GLYPH = node.id'glyph'
5576   local KASHIDA = Babel.attr_kashida
5577   local LOCALE = Babel.attr_locale
5578
5579   if line == nil then
5580     line = {}
5581     line.glue_sign = 1
5582     line.glue_order = 0
5583     line.head = head
5584     line.shift = 0
5585     line.width = size
5586   end
5587
5588   % Exclude last line. todo. But-- it discards one-word lines, too!
5589   % ? Look for glue = 12:15
5590   if (line.glue_sign == 1 and line.glue_order == 0) then
5591     elongs = {}      % Stores elongated candidates of each line
5592     k_list = {}      % And all letters with kashida
5593     pos_inline = 0   % Not yet used
5594
5595     for n in node.traverse_id(GLYPH, line.head) do
5596       pos_inline = pos_inline + 1 % To find where it is. Not used.
5597
5598       % Elongated glyphs
5599       if elong_map then
5600         local locale = node.get_attribute(n, LOCALE)
5601         if elong_map[locale] and elong_map[locale][n.font] and
5602           elong_map[locale][n.font][n.char] then
5603           table.insert(elongs, {node = n, locale = locale} )
5604           node.set_attribute(n.prev, KASHIDA, 0)
5605         end
5606       end
5607
5608       % Tatwil
5609       if Babel.kashida_wts then

```

```

5610         local k_wt = node.get_attribute(n, KASHIDA)
5611         if k_wt > 0 then % todo. parameter for multi inserts
5612             table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5613         end
5614     end
5615
5616 end % of node.traverse_id
5617
5618 if #elongs == 0 and #k_list == 0 then goto next_line end
5619 full = line.width
5620 shift = line.shift
5621 goal = full * Babel.arabic.justify_factor % A bit crude
5622 width = node.dimensions(line.head) % The 'natural' width
5623
5624 % == Elongated ==
5625 % Original idea taken from 'chickenize'
5626 while (#elongs > 0 and width < goal) do
5627     subst_done = true
5628     local x = #elongs
5629     local curr = elongs[x].node
5630     local oldchar = curr.char
5631     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5632     width = node.dimensions(line.head) % Check if the line is too wide
5633     % Substitute back if the line would be too wide and break:
5634     if width > goal then
5635         curr.char = oldchar
5636         break
5637     end
5638     % If continue, pop the just substituted node from the list:
5639     table.remove(elongs, x)
5640 end
5641
5642 % == Tatwil ==
5643 if #k_list == 0 then goto next_line end
5644
5645 width = node.dimensions(line.head) % The 'natural' width
5646 k_curr = #k_list
5647 wt_pos = 1
5648
5649 while width < goal do
5650     subst_done = true
5651     k_item = k_list[k_curr].node
5652     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5653         d = node.copy(k_item)
5654         d.char = 0x0640
5655         line.head, new = node.insert_after(line.head, k_item, d)
5656         width_new = node.dimensions(line.head)
5657         if width > goal or width == width_new then
5658             node.remove(line.head, new) % Better compute before
5659             break
5660         end
5661         width = width_new
5662     end
5663     if k_curr == 1 then
5664         k_curr = #k_list
5665         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5666     else
5667         k_curr = k_curr - 1
5668     end
5669 end
5670
5671 ::next_line::
5672

```

```

5673 % Must take into account marks and ins, see luatex manual.
5674 % Have to be executed only if there are changes. Investigate
5675 % what's going on exactly.
5676 if subst_done and not gc then
5677     d = node.hpack(line.head, full, 'exactly')
5678     d.shift = shift
5679     node.insert_before(head, line, d)
5680     node.remove(head, line)
5681 end
5682 end % if process line
5683 end
5684 }
5685 \endgroup
5686 \fi\fi % Arabic just block

```

## 9.8 Common stuff

```

5687 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5688 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5689 \DisableBabelHook{babel-fontspec}
5690 <<Font selection>>

```

## 9.9 Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a short function which just traverse the node list to carry out the replacements. The table `loc_to_scr` gets the locale form a script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the `\language` and the `\localeid` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with `/` maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```

5691 % TODO - to a lua file
5692 \directlua{
5693 Babel.script_blocks = {
5694   ['dflt'] = {},
5695   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5696             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5697   ['Armn'] = {{0x0530, 0x058F}},
5698   ['Beng'] = {{0x0980, 0x09FF}},
5699   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5700   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5701   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5702             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5703   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5704   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5705             {0xAB00, 0xAB2F}},
5706   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5707   % Don't follow strictly Unicode, which places some Coptic letters in
5708   % the 'Greek and Coptic' block
5709   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5710   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5711             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5712             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5713             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5714             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5715             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5716   ['Hebr'] = {{0x0590, 0x05FF}},
5717   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5718             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5719   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5720   ['Knda'] = {{0x0C80, 0x0CFF}},
5721   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5722             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5723             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},

```

```

5724 ['Lao'] = {{0x0E80, 0x0EFF}},
5725 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5726             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5727             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5728 ['Mahj'] = {{0x11150, 0x1117F}},
5729 ['Mlym'] = {{0x0D00, 0x0D7F}},
5730 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5731 ['Orya'] = {{0x0B00, 0x0B7F}},
5732 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5733 ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5734 ['Taml'] = {{0x0B80, 0x0BFF}},
5735 ['Telu'] = {{0x0C00, 0x0C7F}},
5736 ['Tfng'] = {{0x2D30, 0x2D7F}},
5737 ['Thai'] = {{0x0E00, 0x0E7F}},
5738 ['Tibt'] = {{0x0F00, 0x0FFF}},
5739 ['Vaii'] = {{0xA500, 0xA63F}},
5740 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5741 }
5742
5743 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5744 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5745 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5746
5747 function Babel.locale_map(head)
5748   if not Babel.locale_mapped then return head end
5749
5750   local LOCALE = Babel.attr_locale
5751   local GLYPH = node.id('glyph')
5752   local inmath = false
5753   local toloc_save
5754   for item in node.traverse(head) do
5755     local toloc
5756     if not inmath and item.id == GLYPH then
5757       % Optimization: build a table with the chars found
5758       if Babel.chr_to_loc[item.char] then
5759         toloc = Babel.chr_to_loc[item.char]
5760       else
5761         for lc, maps in pairs(Babel.loc_to_scr) do
5762           for _, rg in pairs(maps) do
5763             if item.char >= rg[1] and item.char <= rg[2] then
5764               Babel.chr_to_loc[item.char] = lc
5765               toloc = lc
5766               break
5767             end
5768           end
5769         end
5770       end
5771       % Now, take action, but treat composite chars in a different
5772       % fashion, because they 'inherit' the previous locale. Not yet
5773       % optimized.
5774       if not toloc and
5775         (item.char >= 0x0300 and item.char <= 0x036F) or
5776         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5777         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5778         toloc = toloc_save
5779       end
5780       if toloc and Babel.locale_props[toloc] and
5781         Babel.locale_props[toloc].letters and
5782         tex.getcatcode(item.char) \string~= 11 then
5783         toloc = nil
5784       end
5785       if toloc and toloc > -1 then
5786         if Babel.locale_props[toloc].lg then

```



```

5787         item.lang = Babel.locale_props[toloc].lg
5788         node.set_attribute(item, LOCALE, toloc)
5789     end
5790     if Babel.locale_props[toloc]['/'..item.font] then
5791         item.font = Babel.locale_props[toloc]['/'..item.font]
5792     end
5793     toloc_save = toloc
5794 end
5795 elseif not inmath and item.id == 7 then % Apply recursively
5796     item.replace = item.replace and Babel.locale_map(item.replace)
5797     item.pre      = item.pre and Babel.locale_map(item.pre)
5798     item.post     = item.post and Babel.locale_map(item.post)
5799 elseif item.id == node.id'math' then
5800     inmath = (item.subtype == 0)
5801 end
5802 end
5803 return head
5804 end
5805 }

```

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

```

5806 \newcommand\babelcharproperty[1]{%
5807   \count@=#1\relax
5808   \ifvmode
5809     \expandafter\bbl@chprop
5810   \else
5811     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5812               vertical mode (preamble or between paragraphs)}%
5813     {See the manual for futher info}%
5814   \fi}
5815 \newcommand\bbl@chprop[3][\the\count@]{%
5816   \@tempcnta=#1\relax
5817   \bbl@ifunset{\bbl@chprop@#2}%
5818   {\bbl@error{No property named '#2'. Allowed values are\\%
5819             direction (bc), mirror (bmg), and linebreak (lb)}%
5820    {See the manual for futher info}}%
5821   {%
5822   \loop
5823     \bbl@cs{\chprop@#2}{#3}%
5824     \ifnum\count@<\@tempcnta
5825       \advance\count@\@ne
5826     \repeat}
5827 \def\bbl@chprop@direction#1{%
5828   \directlua{
5829     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5830     Babel.characters[\the\count@]['d'] = '#1'
5831   }}
5832 \let\bbl@chprop@bc\bbl@chprop@direction
5833 \def\bbl@chprop@mirror#1{%
5834   \directlua{
5835     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5836     Babel.characters[\the\count@]['m'] = '\number#1'
5837   }}
5838 \let\bbl@chprop@bmg\bbl@chprop@mirror
5839 \def\bbl@chprop@linebreak#1{%
5840   \directlua{
5841     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5842     Babel.cjk_characters[\the\count@]['c'] = '#1'
5843   }}
5844 \let\bbl@chprop@lb\bbl@chprop@linebreak
5845 \def\bbl@chprop@locale#1{%
5846   \directlua{

```

```

5847 Babel.chr_to_loc = Babel.chr_to_loc or {}
5848 Babel.chr_to_loc[\the\count@] =
5849 \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
5850 }}

```

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.

```

5851 \directlua{
5852 Babel.nohyphenation = \the\l@nohyphenation
5853 }

```

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).

```

5854 \begingroup
5855 \catcode`\~ = 12
5856 \catcode`\% = 12
5857 \catcode`\& = 14
5858 \catcode`\| = 12
5859 \gdef\babelprehyphenation{&%
5860 \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}}
5861 \gdef\babelposthyphenation{&%
5862 \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}}
5863 \gdef\bbl@postlinebreak{\bbl@settransform{2}[]} &% WIP
5864 \gdef\bbl@settransform#1[#2]#3#4#5{&%
5865 \ifcase#1
5866 \bbl@activateprehyphen
5867 \or
5868 \bbl@activateposthyphen
5869 \fi
5870 \begingroup
5871 \def\babeltempa{\bbl@add@list\babeltempb}&%
5872 \let\babeltempb\@empty
5873 \def\bbl@tempa{#5}&%
5874 \bbl@replace\bbl@tempa{,}{,}&% TODO. Ugly trick to preserve {}
5875 \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5876 \bbl@ifsamestring{##1}{remove}&%
5877 {\bbl@add@list\babeltempb{nil}}}&%
5878 {\directlua{
5879 local rep = {[##1]=}
5880 rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5881 rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5882 rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5883 if #1 == 0 or #1 == 2 then
5884 rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5885 'space = {' .. '%2, %3, %4' .. '}')
5886 rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5887 'spacefactor = {' .. '%2, %3, %4' .. '}')
5888 rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
5889 else
5890 rep = rep:gsub('(no)%s*=%s*([^\s,]*)', Babel.capture_func)
5891 rep = rep:gsub('(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5892 rep = rep:gsub('(post)%s*=%s*([^\s,]*)', Babel.capture_func)
5893 end
5894 tex.print([[string\babeltempa{}} .. rep .. [{}]])
5895 }}}&%
5896 \bbl@foreach\babeltempb{&%
5897 \bbl@forkv{##1}{&%

```

```

5898 \in{,####1,}{,nil,step,data,remove,insert,string,no,pre,&%
5899 no,post,penalty,kashida,space,spacefactor,}&%
5900 \ifin@ \else
5901 \bbl@error
5902 {Bad option '####1' in a transform.\\&%
5903 I'll ignore it but expect more errors}&%
5904 {See the manual for further info.}&%
5905 \fi}&%
5906 \let\bbl@kv@attribute\relax
5907 \let\bbl@kv@label\relax
5908 \let\bbl@kv@fonts\@empty
5909 \bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}&%
5910 \ifx\bbl@kv@fonts\@empty\else\bbl@settransfont\fi
5911 \ifx\bbl@kv@attribute\relax
5912 \ifx\bbl@kv@label\relax\else
5913 \bbl@exp{\bbl@trim@def\bbl@kv@fonts{\bbl@kv@fonts}}&%
5914 \bbl@replace\bbl@kv@fonts{ }{,}&%
5915 \edef\bbl@kv@attribute{\bbl@ATR@\bbl@kv@label @#3@\bbl@kv@fonts}&%
5916 \count@ \z@
5917 \def\bbl@elt##1##2##3{&%
5918 \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
5919 {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
5920 {\count@\@ne}&%
5921 {\bbl@error
5922 {Transforms cannot be re-assigned to different\\&%
5923 fonts. The conflict is in '\bbl@kv@label'.\\&%
5924 Apply the same fonts or use a different label}&%
5925 {See the manual for further details.}}}&%
5926 {}}&%
5927 \bbl@transfont@list
5928 \ifnum\count@=\z@
5929 \bbl@exp{\global\bbl@add\bbl@transfont@list
5930 {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
5931 \fi
5932 \bbl@ifunset{\bbl@kv@attribute}&%
5933 {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
5934 {}}&%
5935 \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
5936 \fi
5937 \else
5938 \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
5939 \fi
5940 \directlua{
5941 local lbkr = Babel.linebreaking.replacements[#1]
5942 local u = unicode.utf8
5943 local id, attr, label
5944 if #1 == 0 or #1 == 2 then
5945 id = \the\csname bbl@id@#3\endcsname\space
5946 else
5947 id = \the\csname l@#3\endcsname\space
5948 end
5949 \ifx\bbl@kv@attribute\relax
5950 attr = -1
5951 \else
5952 attr = luatexbase.registernumber'\bbl@kv@attribute'
5953 \fi
5954 \ifx\bbl@kv@label\relax\else &% Same refs:
5955 label = [==[\bbl@kv@label]==]
5956 \fi
5957 &% Convert pattern:
5958 local patt = string.gsub([==[#4]==], '%s', '')
5959 if #1 == 0 or #1 == 2 then
5960 patt = string.gsub(patt, '|', ' ')

```

```

5961     end
5962     if not u.find(patt, '()', nil, true) then
5963         patt = '()' .. patt .. '()'
5964     end
5965     if #1 == 1 then
5966         patt = string.gsub(patt, '%(%)%^', '^()')
5967         patt = string.gsub(patt, '%$%(%)', '()$')
5968     end
5969     patt = u.gsub(patt, '{(.)}',
5970         function (n)
5971             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5972         end)
5973     patt = u.gsub(patt, '{(%x%x%x%x+)}',
5974         function (n)
5975             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
5976         end)
5977     lbkr[id] = lbkr[id] or {}
5978     table.insert(lbkr[id],
5979         { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
5980 }&%
5981 \endgroup}
5982 \endgroup
5983 \let\bbl@transfont@list@empty
5984 \def\bbl@settransfont{%
5985     \global\let\bbl@settransfont\relax % Execute only once
5986     \gdef\bbl@transfont{%
5987         \def\bbl@elt####1####2####3{%
5988             \bbl@ifblank{####3}%
5989                 {\count@tw@}% Do nothing if no fonts
5990                 {\count@z@
5991                     \bbl@vforeach{####3}{%
5992                         \def\bbl@tempd{#####1}%
5993                         \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
5994                         \ifx\bbl@tempd\bbl@tempe
5995                             \count@ne
5996                         \else\ifx\bbl@tempd\bbl@transfam
5997                             \count@ne
5998                         \fi\fi}%
5999                     \ifcase\count@
6000                         \bbl@csarg\unsetattribute{ATR@####2@####1@####3}%
6001                     \or
6002                         \bbl@csarg\setattribute{ATR@####2@####1@####3}\@ne
6003                     \fi}}%
6004         \bbl@transfont@list}%
6005     \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6006     \gdef\bbl@transfam{-unknown-}%
6007     \bbl@foreach\bbl@font@fams{%
6008         \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6009         \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6010         {\xdef\bbl@transfam{##1}}%
6011     }}}}
6012 \DeclareRobustCommand\enablelocaletransform[1]{%
6013     \bbl@ifunset{\bbl@ATR@#1@\languagename @}%
6014     {\bbl@error
6015         {'#1' for '\languagename' cannot be enabled.\%
6016         Maybe there is a typo or it's a font-dependent transform}%
6017         {See the manual for further details.}}%
6018     {\bbl@csarg\setattribute{ATR@#1@\languagename @}\@ne}}
6019 \DeclareRobustCommand\disablelocaletransform[1]{%
6020     \bbl@ifunset{\bbl@ATR@#1@\languagename @}%
6021     {\bbl@error
6022         {'#1' for '\languagename' cannot be disabled.\%
6023         Maybe there is a typo or it's a font-dependent transform}%

```

```

6024     {See the manual for further details.}}%
6025     {\bbl@csarg\unsetAttribute{ATR@#1@\language @}}
6026 \def\bbl@activateposthyphen{%
6027   \let\bbl@activateposthyphen\relax
6028   \directlua{
6029     require('babel-transforms.lua')
6030     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6031   }}
6032 \def\bbl@activateprehyphen{%
6033   \let\bbl@activateprehyphen\relax
6034   \directlua{
6035     require('babel-transforms.lua')
6036     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6037   }}

```

## 9.10 Bidi

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

```

6038 \def\bbl@activate@preotf{%
6039   \let\bbl@activate@preotf\relax % only once
6040   \directlua{
6041     Babel = Babel or {}
6042     %
6043     function Babel.pre_otfload_v(head)
6044       if Babel.numbers and Babel.digits_mapped then
6045         head = Babel.numbers(head)
6046       end
6047       if Babel.bidi_enabled then
6048         head = Babel.bidi(head, false, dir)
6049       end
6050       return head
6051     end
6052     %
6053     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6054       if Babel.numbers and Babel.digits_mapped then
6055         head = Babel.numbers(head)
6056       end
6057       if Babel.bidi_enabled then
6058         head = Babel.bidi(head, false, dir)
6059       end
6060       return head
6061     end
6062     %
6063     luatexbase.add_to_callback('pre_linebreak_filter',
6064       Babel.pre_otfload_v,
6065       'Babel.pre_otfload_v',
6066     luatexbase.priority_in_callback('pre_linebreak_filter',
6067       'luaotfload.node_processor') or nil)
6068     %
6069     luatexbase.add_to_callback('hpack_filter',
6070       Babel.pre_otfload_h,
6071       'Babel.pre_otfload_h',
6072     luatexbase.priority_in_callback('hpack_filter',
6073       'luaotfload.node_processor') or nil)
6074   }}

```

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=`.

```

6075 \ifnum\bbl@bidimode>\@ne % Excludes default=1
6076   \let\bbl@beforeforeign\leavevmode

```

```

6077 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6078 \RequirePackage{luatexbase}
6079 \bbl@activate@preotf
6080 \directlua{
6081     require('babel-data-bidi.lua')
6082     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6083         require('babel-bidi-basic.lua')
6084     \or
6085         require('babel-bidi-basic-r.lua')
6086     \fi}
6087 \newattribute\bbl@attr@dir
6088 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6089 \bbl@exp{\output{\bodydir\pagedir\the\output}}
6090 \fi
6091 \chardef\bbl@thetextdir\z@
6092 \chardef\bbl@thepardir\z@
6093 \def\bbl@getluadir#1{%
6094     \directlua{
6095         if tex.#1dir == 'TLT' then
6096             tex.sprint('0')
6097         elseif tex.#1dir == 'TRT' then
6098             tex.sprint('1')
6099         end}}
6100 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6101     \ifcase#3\relax
6102         \ifcase\bbl@getluadir{#1}\relax\else
6103             #2 TLT\relax
6104         \fi
6105     \else
6106         \ifcase\bbl@getluadir{#1}\relax
6107             #2 TRT\relax
6108         \fi
6109     \fi}
6110 % ..00PPTT, with masks 0xC (par dir) and 0x3 (text dir)
6111 \def\bbl@thedir{0}
6112 \def\bbl@textdir#1{%
6113     \bbl@setluadir{text}\textdir{#1}%
6114     \chardef\bbl@thetextdir#1\relax
6115     \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6116     \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6117 \def\bbl@pardir#1{% Used twice
6118     \bbl@setluadir{par}\pardir{#1}%
6119     \chardef\bbl@thepardir#1\relax}
6120 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}% Used once
6121 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}% Unused
6122 \def\bbl@dirparastext{\pardir\the\textdir\relax}% Used once

```

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

```

6123 \ifnum\bbl@bidimode>\z@
6124     \def\bbl@insidemath{0}%
6125     \def\bbl@everymath{\def\bbl@insidemath{1}}
6126     \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6127     \frozen@everymath\expandafter{%
6128         \expandafter\bbl@everymath\the\frozen@everymath}
6129     \frozen@everydisplay\expandafter{%
6130         \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6131 \AtBeginDocument{
6132     \directlua{
6133         function Babel.math_box_dir(head)
6134             if not (token.get_macro('bbl@insidemath') == '0') then
6135                 if Babel.hlist_has_bidi(head) then
6136                     local d = node.new(node.id'dir')

```

```

6137         d.dir = '+TRT'
6138         node.insert_before(head, node.has_glyph(head), d)
6139         for item in node.traverse(head) do
6140             node.set_attribute(item,
6141                 Babel.attr_dir, token.get_macro('bbl@thedir'))
6142         end
6143     end
6144 end
6145 return head
6146 end
6147 luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6148     "Babel.math_box_dir", 0)
6149 }}%
6150 \fi

```

## 9.11 Layout

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

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

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

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

```

6151 \bbl@trace{Redefinitions for bidi layout}
6152 %
6153 <<(*More package options)>> ≡
6154 \chardef\bbl@eqnpos\z@
6155 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6156 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6157 <</More package options>>
6158 %
6159 \ifnum\bbl@bidimode>\z@
6160   \ifx\matheqdirmode\undefined\else
6161     \matheqdirmode\@ne % A luatex primitive
6162   \fi
6163   \let\bbl@eqnodir\relax
6164   \def\bbl@eqdel{()}
6165   \def\bbl@eqnum{%
6166     {\normalfont\normalcolor
6167       \expandafter\@firstoftwo\bbl@eqdel
6168       \theequation
6169       \expandafter\@secondoftwo\bbl@eqdel}}
6170   \def\bbl@puteqno#1{\eqno\hbox{#1}}
6171   \def\bbl@putleqno#1{\leqno\hbox{#1}}
6172   \def\bbl@eqno@flip#1{%
6173     \ifdim\predisplaysize=-\maxdimen
6174       \eqno
6175       \hb@xt@.01pt{\hb@xt@\displaywidth{\hss{#1}}\hss}%
6176     \else
6177       \leqno\hbox{#1}%
6178     \fi}
6179   \def\bbl@leqno@flip#1{%
6180     \ifdim\predisplaysize=-\maxdimen
6181       \leqno
6182       \hb@xt@.01pt{\hss\hb@xt@\displaywidth{\hss{#1}}\hss}%
6183     \else
6184       \eqno\hbox{#1}%

```

```

6185 \fi}
6186 \AtBeginDocument{%
6187 \ifx\bb1@noamsmath\relax\else
6188 \ifx\maketag@@@% undefined % Normal equation, eqnarray
6189 \AddToHook{env/equation/begin}{%
6190 \ifnum\bb1@thetextdir>\z@
6191 \def\bb1@mathboxdir{\def\bb1@insidemath{1}}%
6192 \let\@eqnnum\bb1@eqnum
6193 \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6194 \chardef\bb1@thetextdir\z@
6195 \bb1@add\normalfont{\bb1@eqnodir}%
6196 \ifcase\bb1@eqnpos
6197 \let\bb1@puteqno\bb1@eqno@flip
6198 \or
6199 \let\bb1@puteqno\bb1@leqno@flip
6200 \fi
6201 \fi}%
6202 \ifnum\bb1@eqnpos=\tw@% else
6203 \def\endequation{\bb1@puteqno{\@eqnnum}$\@ignoretrue}%
6204 \fi
6205 \AddToHook{env/eqnarray/begin}{%
6206 \ifnum\bb1@thetextdir>\z@
6207 \def\bb1@mathboxdir{\def\bb1@insidemath{1}}%
6208 \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6209 \chardef\bb1@thetextdir\z@
6210 \bb1@add\normalfont{\bb1@eqnodir}%
6211 \ifnum\bb1@eqnpos=\@ne
6212 \def\@eqnnum{%
6213 \setbox\z@\hbox{\bb1@eqnum}%
6214 \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6215 \else
6216 \let\@eqnnum\bb1@eqnum
6217 \fi
6218 \fi}
6219 % Hack. YA luatex bug?:
6220 \expandafter\bb1@sreplace\csname] \endcsname{$$}{\eqno\kern.001pt$}$%
6221 \else % amstex
6222 \bb1@exp{% Hack to hide maybe undefined conditionals:
6223 \chardef\bb1@eqnpos=0%
6224 \<iftagsleft@>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6225 \ifnum\bb1@eqnpos=\@ne
6226 \let\bb1@ams@lap\hbox
6227 \else
6228 \let\bb1@ams@lap\llap
6229 \fi
6230 \ExplSyntaxOn % Required by \bb1@sreplace with \intertext@
6231 \bb1@sreplace\intertext@{\normalbaselines}%
6232 {\normalbaselines
6233 \ifx\bb1@eqnodir\relax\else\bb1@pardir\@ne\bb1@eqnodir\fi}%
6234 \ExplSyntaxOff
6235 \def\bb1@ams@tagbox#1#2{#1{\bb1@eqnodir#2}}% #1=hbox|@lap|flip
6236 \ifx\bb1@ams@lap\hbox % leqno
6237 \def\bb1@ams@flip#1{%
6238 \hbox to 0.01pt{\hss\hbox to\displaywidth{{#1}\hss}}}%
6239 \else % eqno
6240 \def\bb1@ams@flip#1{%
6241 \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}\hss}}%
6242 \fi
6243 \def\bb1@ams@preset#1{%
6244 \def\bb1@mathboxdir{\def\bb1@insidemath{1}}%
6245 \ifnum\bb1@thetextdir>\z@
6246 \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6247 \bb1@sreplace\textdef@{\hbox}{\bb1@ams@tagbox\hbox}%

```



```

6248      \bbl@sreplace\maketag@@@{\hbox}{\bbl@ams@tagbox#1}%
6249      \fi}%
6250      \ifnum\bbl@eqnpos=\tw@\else
6251      \def\bbl@ams@equation{%
6252      \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6253      \ifnum\bbl@thetextdir>\z@
6254      \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6255      \chardef\bbl@thetextdir\z@
6256      \bbl@add\normalfont{\bbl@eqnodir}%
6257      \ifcase\bbl@eqnpos
6258      \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
6259      \or
6260      \def\veqno##1##2{\bbl@leqno@flip{##1##2}}%
6261      \fi
6262      \fi}%
6263      \AddToHook{env/equation/begin}{\bbl@ams@equation}%
6264      \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6265      \fi
6266      \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6267      \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6268      \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6269      \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6270      \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6271      \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6272      \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6273      % Hackish, for proper alignment. Don't ask me why it works!:
6274      \bbl@exp{% Avoid a 'visible' conditional
6275      \\\AddToHook{env/align*/end}{\<iftag>\<else>\\tag*{\<fi>}}%
6276      \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6277      \AddToHook{env/split/before}{%
6278      \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6279      \ifnum\bbl@thetextdir>\z@
6280      \bbl@ifsamestring\@currentenv{equation}%
6281      {\ifx\bbl@ams@lap\hbox % leqno
6282      \def\bbl@ams@flip#1{%
6283      \hbox to 0.01pt{\hbox to\displaywidth{##1}\hss}\hss}}%
6284      \else
6285      \def\bbl@ams@flip#1{%
6286      \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{##1}}}%
6287      \fi}%
6288      }%
6289      \fi}%
6290      \fi\fi}
6291 \fi
6292 \def\bbl@provide@extra#1{%
6293 % == Counters: mapdigits ==
6294 % Native digits
6295 \ifx\bbl@KVP@mapdigits\@nnil\else
6296 \bbl@ifunset{\bbl@dgnat@language}{%
6297 {\RequirePackage{luatexbase}%
6298 \bbl@activate@preotf
6299 \directlua{
6300 Babel = Babel or {} %%% -> presets in luababel
6301 Babel.digits_mapped = true
6302 Babel.digits = Babel.digits or {}
6303 Babel.digits[\the\localeid] =
6304 table.pack(string.utfvalue('\bbl@cl{dgnat}'))
6305 if not Babel.numbers then
6306 function Babel.numbers(head)
6307 local LOCALE = Babel.attr_locale
6308 local GLYPH = node.id'glyph'
6309 local inmath = false
6310 for item in node.traverse(head) do

```

```

6311         if not inmath and item.id == GLYPH then
6312             local temp = node.get_attribute(item, LOCALE)
6313             if Babel.digits[temp] then
6314                 local chr = item.char
6315                 if chr > 47 and chr < 58 then
6316                     item.char = Babel.digits[temp][chr-47]
6317                 end
6318             end
6319             elseif item.id == node.id'math' then
6320                 inmath = (item.subtype == 0)
6321             end
6322         end
6323         return head
6324     end
6325 end
6326 }}%
6327 \fi
6328 % == transforms ==
6329 \ifx\bbl@KVP@transforms\@nnil\else
6330     \def\bbl@elt##1##2##3{%
6331         \in@{$transforms.}{$##1}%
6332         \ifin@
6333             \def\bbl@tempa{##1}%
6334             \bbl@replace\bbl@tempa{transforms.}{}%
6335             \bbl@carg\bbl@transforms{babel\bbl@tempa}{##2}{##3}%
6336         \fi}%
6337     \csname bbl@inidata@\language\endcsname
6338     \bbl@release@transforms\relax % \relax closes the last item.
6339 \fi}
6340 % Start tabular here:
6341 \def\localerestoredirs{%
6342     \ifcase\bbl@thetextdir
6343         \ifnum\textdirection=\z@\else\textdir TLT\fi
6344     \else
6345         \ifnum\textdirection=\@ne\else\textdir TRT\fi
6346     \fi
6347     \ifcase\bbl@thepardir
6348         \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6349     \else
6350         \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6351     \fi}
6352 \IfBabelLayout{tabular}%
6353 { \chardef\bbl@tabular@mode\tw@}% All RTL
6354 { \IfBabelLayout{notabular}%
6355     { \chardef\bbl@tabular@mode\z@}%
6356     { \chardef\bbl@tabular@mode\@ne}% Mixed, with LTR cols
6357 \ifnum\bbl@bidimode>\@ne
6358     \ifnum\bbl@tabular@mode=\@ne
6359         \let\bbl@parabefore\relax
6360         \AddToHook{para/before}{\bbl@parabefore}
6361         \AtBeginDocument{%
6362             \bbl@replace\@tabular{$}{$}%
6363             \def\bbl@insidemath{0}%
6364             \def\bbl@parabefore{\localerestoredirs}}%
6365         \ifnum\bbl@tabular@mode=\@ne
6366             \bbl@ifunset{ @tabclassz}{ }{%
6367                 \bbl@exp{% Hide conditionals
6368                     \\\bbl@sreplace\\ @tabclassz
6369                     {< ifcase>\\ @chnum}%
6370                     {\\ \localerestoredirs< ifcase>\\ @chnum}}}%
6371             \ifpackageloaded{colortbl}%
6372                 {\bbl@sreplace @classz
6373                     {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%

```

```

6374      {\@ifpackageloaded{array}%
6375      {\bbl@exp{% Hide conditionals
6376      \\\bbl@sreplace\\\@classz
6377      {\<ifcase>\\\@chnum}%
6378      {\bgroup\\\localerestoredirs\<ifcase>\\\@chnum}%
6379      \\\bbl@sreplace\\\@classz
6380      {\do@row@strut\<fi>}{\do@row@strut\<fi>\egroup}}}%
6381      }}}%
6382  \fi}
6383 \fi
6384 \AtBeginDocument{%
6385   \@ifpackageloaded{multicol}%
6386   {\toks@ \expandafter{\multicolumn@out}%
6387   \edef\multicolumn@out{\bodydir\pagedir\the\toks@}}%
6388   {}%
6389 \fi
6390 \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.

6391 \ifnum\bbl@bidimode>\z@
6392   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6393     \bbl@exp{%
6394       \def\\\bbl@insidemath{0}%
6395       \mathdir\the\bodydir
6396       #1% Once entered in math, set boxes to restore values
6397       \<ifmmode>%
6398       \everyvbox{%
6399         \the\everyvbox
6400         \bodydir\the\bodydir
6401         \mathdir\the\mathdir
6402         \everyhbox{\the\everyhbox}%
6403         \everyvbox{\the\everyvbox}}%
6404       \everyhbox{%
6405         \the\everyhbox
6406         \bodydir\the\bodydir
6407         \mathdir\the\mathdir
6408         \everyhbox{\the\everyhbox}%
6409         \everyvbox{\the\everyvbox}}%
6410       \<fi>}}%
6411   \def\@hangfrom#1{%
6412     \setbox\@tempboxa\hbox{#1}%
6413     \hangindent\wd\@tempboxa
6414     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6415       \shapemode\@ne
6416       \fi
6417     \noindent\box\@tempboxa}
6418 \fi
6419 \IfBabelLayout{tabular}
6420 {\let\bbl@OL@tabular\@tabular
6421  \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6422  \let\bbl@NL@tabular\@tabular
6423  \AtBeginDocument{%
6424    \ifx\bbl@NL@tabular\@tabular\else
6425      \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6426      \let\bbl@NL@tabular\@tabular
6427    \fi}}
6428 {}
6429 \IfBabelLayout{lists}
6430 {\let\bbl@OL@list\list
6431  \bbl@sreplace\list{\parshape}{\bbl@listparshape}%

```

```

6432 \let\bbl@NL@list\list
6433 \def\bbl@listparshape#1#2#3{%
6434   \parshape #1 #2 #3 %
6435   \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6436     \shapemode\tw@
6437   \fi}}
6438 {}
6439 \IfBabelLayout{graphics}
6440 {\let\bbl@pictresetdir\relax
6441  \def\bbl@pictsetdir#1{%
6442    \ifcase\bbl@thetextdir
6443      \let\bbl@pictresetdir\relax
6444    \else
6445      \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6446        \or\textdir TLT
6447        \else\bodydir TLT \textdir TLT
6448      \fi
6449      % \(\text|par)dir required in pgf:
6450      \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6451    \fi}%
6452 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6453 \directlua{
6454   Babel.get_picture_dir = true
6455   Babel.picture_has_bidi = 0
6456   %
6457   function Babel.picture_dir (head)
6458     if not Babel.get_picture_dir then return head end
6459     if Babel.hlist_has_bidi(head) then
6460       Babel.picture_has_bidi = 1
6461     end
6462     return head
6463   end
6464   luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6465     "Babel.picture_dir")
6466 }%
6467 \AtBeginDocument{%
6468   \def\LS@rot{%
6469     \setbox\@outputbox\vbox{%
6470       \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
6471   \long\def\put(#1,#2)#3{%
6472     \@killglue
6473     % Try:
6474     \ifx\bbl@pictresetdir\relax
6475       \def\bbl@tempc{0}%
6476     \else
6477       \directlua{
6478         Babel.get_picture_dir = true
6479         Babel.picture_has_bidi = 0
6480       }%
6481       \setbox\z@\hb@xt@\z@{%
6482         \@defaultunitsset\@tempdimc{#1}\unitlength
6483         \kern\@tempdimc
6484         #3\hss}% TODO: #3 executed twice (below). That's bad.
6485       \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6486     \fi
6487     % Do:
6488     \@defaultunitsset\@tempdimc{#2}\unitlength
6489     \raise\@tempdimc\hb@xt@\z@{%
6490       \@defaultunitsset\@tempdimc{#1}\unitlength
6491       \kern\@tempdimc
6492       {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
6493     \ignorespaces}%
6494   \MakeRobust\put}%

```

```

6495 \AtBeginDocument
6496   {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir\gobble}%
6497   \ifx\pgfpicture\undefined\else % TODO. Allow deactivate?
6498     \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6499     \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6500     \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6501   \fi
6502   \ifx\tikzpicture\undefined\else
6503     \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\tw}%
6504     \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6505     \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw}%
6506   \fi
6507   \ifx\tcolorbox\undefined\else
6508     \def\tcb@drawing@env@begin{%
6509       \csname tcb@before@\tcb@split@state\endcsname
6510       \bbl@pictsetdir\tw@
6511       \begin{\kvtcb@graphenv}%
6512       \tcb@bbdraw%
6513       \tcb@apply@graph@patches
6514     }%
6515     \def\tcb@drawing@env@end{%
6516       \end{\kvtcb@graphenv}%
6517       \bbl@pictresetdir
6518       \csname tcb@after@\tcb@split@state\endcsname
6519     }%
6520   \fi
6521   }}
6522 {}

```

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.

```

6523 \IfBabelLayout{counters*}%
6524   {\bbl@add\bbl@opt@layout{.counters.}%
6525   \directlua{
6526     luatexbase.add_to_callback("process_output_buffer",
6527       Babel.discard_sublr , "Babel.discard_sublr") }%
6528   }}
6529 \IfBabelLayout{counters}%
6530   {\let\bbl@OL@@@textsuperscript\@textsuperscript
6531   \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6532   \let\bbl@latinarabic=\@arabic
6533   \let\bbl@OL@@@arabic\@arabic
6534   \def\@arabic#1{\bbl@sublr{\bbl@latinarabic#1}}%
6535   \@ifpackagewith{babel}{bidi=default}%
6536   {\let\bbl@asciroman=\@roman
6537   \let\bbl@OL@@@roman\@roman
6538   \def\@roman#1{\bbl@sublr{\ensureascii{\bbl@asciroman#1}}}%
6539   \let\bbl@asciiRoman=\@Roman
6540   \let\bbl@OL@@@roman\@Roman
6541   \def\@Roman#1{\bbl@sublr{\ensureascii{\bbl@asciiRoman#1}}}%
6542   \let\bbl@OL@labelenumii\labelenumii
6543   \def\labelenumii{\theenumii}%
6544   \let\bbl@OL@p@enumiii\p@enumiii
6545   \def\p@enumiii{\p@enumii}\theenumii{}}{}
6546 <<Footnote changes>>
6547 \IfBabelLayout{footnotes}%
6548   {\let\bbl@OL@footnote\footnote
6549   \BabelFootnote\footnote\languagename{}}%
6550   \BabelFootnote\localfootnote\languagename{}}%
6551   \BabelFootnote\mainfootnote{}}{}
6552 {}

```

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.

```

6553 \IfBabelLayout{extras}%
6554   {\let\bbl@OL@underline\underline
6555    \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
6556    \let\bbl@OL@LaTeX2e\LaTeX2e
6557    \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6558      \if b\expandafter\@car\@series\@nil\boldmath\fi
6559      \babelsublr}%
6560      \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
6561   {}
6562 \end{luatex}

```

## 9.12 Lua: transforms

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

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

```

6563 (*transforms)
6564 Babel.linebreaking.replacements = {}
6565 Babel.linebreaking.replacements[0] = {} -- pre
6566 Babel.linebreaking.replacements[1] = {} -- post
6567 Babel.linebreaking.replacements[2] = {} -- post-line WIP
6568
6569 -- Discretionaries contain strings as nodes
6570 function Babel.str_to_nodes(fn, matches, base)
6571   local n, head, last
6572   if fn == nil then return nil end
6573   for s in string.utfvalues(fn(matches)) do
6574     if base.id == 7 then
6575       base = base.replace
6576     end
6577     n = node.copy(base)
6578     n.char = s
6579     if not head then
6580       head = n
6581     else
6582       last.next = n
6583     end
6584     last = n
6585   end
6586   return head
6587 end
6588
6589 Babel.fetch_subtext = {}
6590
6591 Babel.ignore_pre_char = function(node)
6592   return (node.lang == Babel.nohyphenation)
6593 end
6594
6595 -- Merging both functions doesn't seem feasible, because there are too
6596 -- many differences.
6597 Babel.fetch_subtext[0] = function(head)
6598   local word_string = ''
6599   local word_nodes = {}

```

```

6600 local lang
6601 local item = head
6602 local inmath = false
6603
6604 while item do
6605     if item.id == 11 then
6606         inmath = (item.subtype == 0)
6607     end
6608
6609     if inmath then
6610         -- pass
6611     end
6612
6613     elseif item.id == 29 then
6614         local locale = node.get_attribute(item, Babel.attr_locale)
6615
6616         if lang == locale or lang == nil then
6617             lang = lang or locale
6618             if Babel.ignore_pre_char(item) then
6619                 word_string = word_string .. Babel.us_char
6620             else
6621                 word_string = word_string .. unicode.utf8.char(item.char)
6622             end
6623             word_nodes[#word_nodes+1] = item
6624         else
6625             break
6626         end
6627
6628     elseif item.id == 12 and item.subtype == 13 then
6629         word_string = word_string .. ' '
6630         word_nodes[#word_nodes+1] = item
6631
6632         -- Ignore leading unrecognized nodes, too.
6633         elseif word_string ~= '' then
6634             word_string = word_string .. Babel.us_char
6635             word_nodes[#word_nodes+1] = item -- Will be ignored
6636         end
6637
6638     item = item.next
6639 end
6640
6641 -- Here and above we remove some trailing chars but not the
6642 -- corresponding nodes. But they aren't accessed.
6643 if word_string:sub(-1) == ' ' then
6644     word_string = word_string:sub(1,-2)
6645 end
6646 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6647 return word_string, word_nodes, item, lang
6648 end
6649
6650 Babel.fetch_subtext[1] = function(head)
6651     local word_string = ''
6652     local word_nodes = {}
6653     local lang
6654     local item = head
6655     local inmath = false
6656
6657     while item do
6658
6659         if item.id == 11 then
6660             inmath = (item.subtype == 0)
6661         end
6662

```

```

6663     if inmath then
6664         -- pass
6665
6666     elseif item.id == 29 then
6667         if item.lang == lang or lang == nil then
6668             if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6669                 lang = lang or item.lang
6670                 word_string = word_string .. unicode.utf8.char(item.char)
6671                 word_nodes[#word_nodes+1] = item
6672             end
6673         else
6674             break
6675         end
6676
6677     elseif item.id == 7 and item.subtype == 2 then
6678         word_string = word_string .. '='
6679         word_nodes[#word_nodes+1] = item
6680
6681     elseif item.id == 7 and item.subtype == 3 then
6682         word_string = word_string .. '|'
6683         word_nodes[#word_nodes+1] = item
6684
6685     -- (1) Go to next word if nothing was found, and (2) implicitly
6686     -- remove leading USs.
6687     elseif word_string == '' then
6688         -- pass
6689
6690     -- This is the responsible for splitting by words.
6691     elseif (item.id == 12 and item.subtype == 13) then
6692         break
6693
6694     else
6695         word_string = word_string .. Babel.us_char
6696         word_nodes[#word_nodes+1] = item -- Will be ignored
6697     end
6698
6699     item = item.next
6700 end
6701
6702 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6703 return word_string, word_nodes, item, lang
6704 end
6705
6706 function Babel.pre_hyphenate_replace(head)
6707     Babel.hyphenate_replace(head, 0)
6708 end
6709
6710 function Babel.post_hyphenate_replace(head)
6711     Babel.hyphenate_replace(head, 1)
6712 end
6713
6714 Babel.us_char = string.char(31)
6715
6716 function Babel.hyphenate_replace(head, mode)
6717     local u = unicode.utf8
6718     local lbkr = Babel.linebreaking.replacements[mode]
6719     if mode == 2 then mode = 0 end -- WIP
6720
6721     local word_head = head
6722
6723     while true do -- for each subtext block
6724
6725         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)

```



```

6726
6727     if Babel.debug then
6728         print()
6729         print((mode == 0) and '@@@<' or '@@@>', w)
6730     end
6731
6732     if nw == nil and w == '' then break end
6733
6734     if not lang then goto next end
6735     if not lbkr[lang] then goto next end
6736
6737     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6738     -- loops are nested.
6739     for k=1, #lbkr[lang] do
6740         local p = lbkr[lang][k].pattern
6741         local r = lbkr[lang][k].replace
6742         local attr = lbkr[lang][k].attr or -1
6743
6744         if Babel.debug then
6745             print('*****', p, mode)
6746         end
6747
6748         -- This variable is set in some cases below to the first *byte*
6749         -- after the match, either as found by u.match (faster) or the
6750         -- computed position based on sc if w has changed.
6751         local last_match = 0
6752         local step = 0
6753
6754         -- For every match.
6755         while true do
6756             if Babel.debug then
6757                 print('====')
6758             end
6759             local new -- used when inserting and removing nodes
6760
6761             local matches = { u.match(w, p, last_match) }
6762
6763             if #matches < 2 then break end
6764
6765             -- Get and remove empty captures (with ()'s, which return a
6766             -- number with the position), and keep actual captures
6767             -- (from (...)), if any, in matches.
6768             local first = table.remove(matches, 1)
6769             local last = table.remove(matches, #matches)
6770             -- Non re-fetched substrings may contain \31, which separates
6771             -- subsubstrings.
6772             if string.find(w:sub(first, last-1), Babel.us_char) then break end
6773
6774             local save_last = last -- with A()BC()D, points to D
6775
6776             -- Fix offsets, from bytes to unicode. Explained above.
6777             first = u.len(w:sub(1, first-1)) + 1
6778             last = u.len(w:sub(1, last-1)) -- now last points to C
6779
6780             -- This loop stores in a small table the nodes
6781             -- corresponding to the pattern. Used by 'data' to provide a
6782             -- predictable behavior with 'insert' (w_nodes is modified on
6783             -- the fly), and also access to 'remove'd nodes.
6784             local sc = first-1 -- Used below, too
6785             local data_nodes = {}
6786
6787             local enabled = true
6788             for q = 1, last-first+1 do

```

```

6789     data_nodes[q] = w_nodes[sc+q]
6790     if enabled
6791         and attr > -1
6792         and not node.has_attribute(data_nodes[q], attr)
6793     then
6794         enabled = false
6795     end
6796 end
6797
6798 -- This loop traverses the matched substring and takes the
6799 -- corresponding action stored in the replacement list.
6800 -- sc = the position in substr nodes / string
6801 -- rc = the replacement table index
6802 local rc = 0
6803
6804 while rc < last-first+1 do -- for each replacement
6805     if Babel.debug then
6806         print('.....', rc + 1)
6807     end
6808     sc = sc + 1
6809     rc = rc + 1
6810
6811     if Babel.debug then
6812         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6813         local ss = ''
6814         for itt in node.traverse(head) do
6815             if itt.id == 29 then
6816                 ss = ss .. unicode.utf8.char(itt.char)
6817             else
6818                 ss = ss .. '{' .. itt.id .. '}'
6819             end
6820         end
6821         print('*****', ss)
6822     end
6823
6824     local crep = r[rc]
6825     local item = w_nodes[sc]
6826     local item_base = item
6827     local placeholder = Babel.us_char
6828     local d
6829
6830
6831     if crep and crep.data then
6832         item_base = data_nodes[crep.data]
6833     end
6834
6835     if crep then
6836         step = crep.step or 0
6837     end
6838
6839     if (not enabled) or (crep and next(crep) == nil) then -- = {}
6840         last_match = save_last -- Optimization
6841         goto next
6842
6843     elseif crep == nil or crep.remove then
6844         node.remove(head, item)
6845         table.remove(w_nodes, sc)
6846         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6847         sc = sc - 1 -- Nothing has been inserted.
6848         last_match = utf8.offset(w, sc+1+step)
6849         goto next
6850
6851     elseif crep and crep.kashida then -- Experimental

```

```

6852         node.set_attribute(item,
6853             Babel.attr_kashida,
6854             crep.kashida)
6855         last_match = utf8.offset(w, sc+1+step)
6856         goto next
6857
6858     elseif crep and crep.string then
6859         local str = crep.string(matches)
6860         if str == '' then -- Gather with nil
6861             node.remove(head, item)
6862             table.remove(w_nodes, sc)
6863             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6864             sc = sc - 1 -- Nothing has been inserted.
6865         else
6866             local loop_first = true
6867             for s in string.utfvalues(str) do
6868                 d = node.copy(item_base)
6869                 d.char = s
6870                 if loop_first then
6871                     loop_first = false
6872                     head, new = node.insert_before(head, item, d)
6873                     if sc == 1 then
6874                         word_head = head
6875                     end
6876                     w_nodes[sc] = d
6877                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6878                 else
6879                     sc = sc + 1
6880                     head, new = node.insert_before(head, item, d)
6881                     table.insert(w_nodes, sc, new)
6882                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6883                 end
6884                 if Babel.debug then
6885                     print('....', 'str')
6886                     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6887                 end
6888             end -- for
6889             node.remove(head, item)
6890         end -- if ''
6891         last_match = utf8.offset(w, sc+1+step)
6892         goto next
6893
6894     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6895         d = node.new(7, 3) -- (disc, regular)
6896         d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
6897         d.post = Babel.str_to_nodes(crep.post, matches, item_base)
6898         d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6899         d.attr = item_base.attr
6900         if crep.pre == nil then -- TeXbook p96
6901             d.penalty = crep.penalty or tex.hyphenpenalty
6902         else
6903             d.penalty = crep.penalty or tex.exhyphenpenalty
6904         end
6905         placeholder = '|'
6906         head, new = node.insert_before(head, item, d)
6907
6908     elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6909         -- ERROR
6910
6911     elseif crep and crep.penalty then
6912         d = node.new(14, 0) -- (penalty, userpenalty)
6913         d.attr = item_base.attr
6914         d.penalty = crep.penalty

```

```

6915         head, new = node.insert_before(head, item, d)
6916
6917     elseif crep and crep.space then
6918         -- 655360 = 10 pt = 10 * 65536 sp
6919         d = node.new(12, 13) -- (glue, spaceskip)
6920         local quad = font.getfont(item_base.font).size or 655360
6921         node.setglue(d, crep.space[1] * quad,
6922                     crep.space[2] * quad,
6923                     crep.space[3] * quad)
6924         if mode == 0 then
6925             placeholder = ' '
6926         end
6927         head, new = node.insert_before(head, item, d)
6928
6929     elseif crep and crep.spacefactor then
6930         d = node.new(12, 13) -- (glue, spaceskip)
6931         local base_font = font.getfont(item_base.font)
6932         node.setglue(d,
6933                     crep.spacefactor[1] * base_font.parameters['space'],
6934                     crep.spacefactor[2] * base_font.parameters['space_stretch'],
6935                     crep.spacefactor[3] * base_font.parameters['space_shrink'])
6936         if mode == 0 then
6937             placeholder = ' '
6938         end
6939         head, new = node.insert_before(head, item, d)
6940
6941     elseif mode == 0 and crep and crep.space then
6942         -- ERROR
6943
6944     end -- ie replacement cases
6945
6946     -- Shared by disc, space and penalty.
6947     if sc == 1 then
6948         word_head = head
6949     end
6950     if crep.insert then
6951         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6952         table.insert(w_nodes, sc, new)
6953         last = last + 1
6954     else
6955         w_nodes[sc] = d
6956         node.remove(head, item)
6957         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6958     end
6959
6960     last_match = utf8.offset(w, sc+1+step)
6961
6962     ::next::
6963
6964     end -- for each replacement
6965
6966     if Babel.debug then
6967         print('.....', '/')
6968         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6969     end
6970
6971     end -- for match
6972
6973     end -- for patterns
6974
6975     ::next::
6976     word_head = nw
6977     end -- for substring

```

```

6978 return head
6979 end
6980
6981 -- This table stores capture maps, numbered consecutively
6982 Babel.capture_maps = {}
6983
6984 -- The following functions belong to the next macro
6985 function Babel.capture_func(key, cap)
6986   local ret = "[" .. cap:gsub('{{[0-9]}}', "]]..m[%1]..[" .. "]"
6987   local cnt
6988   local u = unicode.utf8
6989   ret, cnt = ret:gsub('{{[0-9]}|(^|+)|(.-)}', Babel.capture_func_map)
6990   if cnt == 0 then
6991     ret = u.gsub(ret, '{{(%x%x%x%x+)}',
6992       function (n)
6993         return u.char(tonumber(n, 16))
6994       end)
6995   end
6996   ret = ret:gsub("%[%[%]]%.%", '')
6997   ret = ret:gsub("%.%.%[%[%]]%", '')
6998   return key .. [[=function(m) return ]] .. ret .. [[ end]]
6999 end
7000
7001 function Babel.capt_map(from, mapno)
7002   return Babel.capture_maps[mapno][from] or from
7003 end
7004
7005 -- Handle the {n|abc|ABC} syntax in captures
7006 function Babel.capture_func_map(capno, from, to)
7007   local u = unicode.utf8
7008   from = u.gsub(from, '{{(%x%x%x%x+)}',
7009     function (n)
7010       return u.char(tonumber(n, 16))
7011     end)
7012   to = u.gsub(to, '{{(%x%x%x%x+)}',
7013     function (n)
7014       return u.char(tonumber(n, 16))
7015     end)
7016   local froms = {}
7017   for s in string.utfcharacters(from) do
7018     table.insert(froms, s)
7019   end
7020   local cnt = 1
7021   table.insert(Babel.capture_maps, {})
7022   local mlen = table.getn(Babel.capture_maps)
7023   for s in string.utfcharacters(to) do
7024     Babel.capture_maps[mlen][froms[cnt]] = s
7025     cnt = cnt + 1
7026   end
7027   return "]]..Babel.capt_map(m[" .. capno .. "], " ..
7028     (mlen) .. ").." .. "[["
7029 end
7030
7031 -- Create/Extend reversed sorted list of kashida weights:
7032 function Babel.capture_kashida(key, wt)
7033   wt = tonumber(wt)
7034   if Babel.kashida_wts then
7035     for p, q in ipairs(Babel.kashida_wts) do
7036       if wt == q then
7037         break
7038       elseif wt > q then
7039         table.insert(Babel.kashida_wts, p, wt)
7040         break

```

```

7041     elseif table.getn(Babel.kashida_wts) == p then
7042         table.insert(Babel.kashida_wts, wt)
7043     end
7044 end
7045 else
7046     Babel.kashida_wts = { wt }
7047 end
7048 return 'kashida = ' .. wt
7049 end
7050 </transforms>

```

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

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

```

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

```

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

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

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

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

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

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

```

7051 (*basic-r)
7052 Babel = Babel or {}
7053
7054 Babel.bidi_enabled = true
7055
7056 require('babel-data-bidi.lua')
7057
7058 local characters = Babel.characters
7059 local ranges = Babel.ranges
7060
7061 local DIR = node.id("dir")
7062
7063 local function dir_mark(head, from, to, outer)
7064     dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse

```

```

7065 local d = node.new(DIR)
7066 d.dir = '+' .. dir
7067 node.insert_before(head, from, d)
7068 d = node.new(DIR)
7069 d.dir = '-' .. dir
7070 node.insert_after(head, to, d)
7071 end
7072
7073 function Babel.bidi(head, ispar)
7074   local first_n, last_n          -- first and last char with nums
7075   local last_es                  -- an auxiliary 'last' used with nums
7076   local first_d, last_d          -- first and last char in L/R block
7077   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):

```

7078 local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7079 local strong_lr = (strong == 'l') and 'l' or 'r'
7080 local outer = strong
7081
7082 local new_dir = false
7083 local first_dir = false
7084 local inmath = false
7085
7086 local last_lr
7087
7088 local type_n = ''
7089
7090 for item in node.traverse(head) do
7091   -- three cases: glyph, dir, otherwise
7092   if item.id == node.id'glyph'
7093     or (item.id == 7 and item.subtype == 2) then
7094     local itemchar
7095     if item.id == 7 and item.subtype == 2 then
7096       itemchar = item.replace.char
7097     else
7098       itemchar = item.char
7099     end
7100     local chardata = characters[itemchar]
7101     dir = chardata and chardata.d or nil
7102     if not dir then
7103       for nn, et in ipairs(ranges) do
7104         if itemchar < et[1] then
7105           break
7106         elseif itemchar <= et[2] then
7107           dir = et[3]
7108           break
7109         end
7110       end
7111     end
7112     dir = dir or 'l'
7113     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).

```

7116   if new_dir then
7117     attr_dir = 0
7118     for at in node.traverse(item.attr) do

```

```

7119         if at.number == Babel.attr_dir then
7120             attr_dir = at.value & 0x3
7121         end
7122     end
7123     if attr_dir == 1 then
7124         strong = 'r'
7125     elseif attr_dir == 2 then
7126         strong = 'al'
7127     else
7128         strong = 'l'
7129     end
7130     strong_lr = (strong == 'l') and 'l' or 'r'
7131     outer = strong_lr
7132     new_dir = false
7133 end
7134
7135 if dir == 'nsm' then dir = strong end -- W1

```

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

```

7136     dir_real = dir -- We need dir_real to set strong below
7137     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:

```

7138     if strong == 'al' then
7139         if dir == 'en' then dir = 'an' end -- W2
7140         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7141         strong_lr = 'r' -- W3
7142     end

```

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

```

7143     elseif item.id == node.id'dir' and not inmath then
7144         new_dir = true
7145         dir = nil
7146     elseif item.id == node.id'math' then
7147         inmath = (item.subtype == 0)
7148     else
7149         dir = nil -- Not a char
7150     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>.

```

7151     if dir == 'en' or dir == 'an' or dir == 'et' then
7152         if dir ~= 'et' then
7153             type_n = dir
7154         end
7155         first_n = first_n or item
7156         last_n = last_es or item
7157         last_es = nil
7158     elseif dir == 'es' and last_n then -- W3+W6
7159         last_es = item
7160     elseif dir == 'cs' then -- it's right - do nothing
7161     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7162         if strong_lr == 'r' and type_n ~= '' then
7163             dir_mark(head, first_n, last_n, 'r')
7164         elseif strong_lr == 'l' and first_d and type_n == 'an' then
7165             dir_mark(head, first_n, last_n, 'r')
7166             dir_mark(head, first_d, last_d, outer)
7167             first_d, last_d = nil, nil
7168         elseif strong_lr == 'l' and type_n ~= '' then
7169             last_d = last_n

```



```

7170     end
7171     type_n = ''
7172     first_n, last_n = nil, nil
7173 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:

```

7174   if dir == 'l' or dir == 'r' then
7175     if dir ~= outer then
7176       first_d = first_d or item
7177       last_d = item
7178     elseif first_d and dir ~= strong_lr then
7179       dir_mark(head, first_d, last_d, outer)
7180       first_d, last_d = nil, nil
7181     end
7182   end

```

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

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

```

7183   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7184     item.char = characters[item.char] and
7185               characters[item.char].m or item.char
7186   elseif (dir or new_dir) and last_lr ~= item then
7187     local mir = outer .. strong_lr .. (dir or outer)
7188     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7189       for ch in node.traverse(node.next(last_lr)) do
7190         if ch == item then break end
7191         if ch.id == node.id'glyph' and characters[ch.char] then
7192           ch.char = characters[ch.char].m or ch.char
7193         end
7194       end
7195     end
7196   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).

```

7197   if dir == 'l' or dir == 'r' then
7198     last_lr = item
7199     strong = dir_real          -- Don't search back - best save now
7200     strong_lr = (strong == 'l') and 'l' or 'r'
7201   elseif new_dir then
7202     last_lr = nil
7203   end
7204 end

```

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

```

7205   if last_lr and outer == 'r' then
7206     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7207       if characters[ch.char] then
7208         ch.char = characters[ch.char].m or ch.char
7209       end
7210     end
7211   end
7212   if first_n then
7213     dir_mark(head, first_n, last_n, outer)
7214   end
7215   if first_d then
7216     dir_mark(head, first_d, last_d, outer)
7217   end

```

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

```
7218 return node.prev(head) or head
7219 end
7220 </basic-r>
```

And here the Lua code for bidi=basic:

```
7221 <*basic>
7222 Babel = Babel or {}
7223
7224 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
7225
7226 Babel.fontmap = Babel.fontmap or {}
7227 Babel.fontmap[0] = {}      -- l
7228 Babel.fontmap[1] = {}      -- r
7229 Babel.fontmap[2] = {}      -- al/an
7230
7231 Babel.bidi_enabled = true
7232 Babel.mirroring_enabled = true
7233
7234 require('babel-data-bidi.lua')
7235
7236 local characters = Babel.characters
7237 local ranges = Babel.ranges
7238
7239 local DIR = node.id('dir')
7240 local GLYPH = node.id('glyph')
7241
7242 local function insert_implicit(head, state, outer)
7243   local new_state = state
7244   if state.sim and state.eim and state.sim ~= state.eim then
7245     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
7246     local d = node.new(DIR)
7247     d.dir = '+' .. dir
7248     node.insert_before(head, state.sim, d)
7249     local d = node.new(DIR)
7250     d.dir = '-' .. dir
7251     node.insert_after(head, state.eim, d)
7252   end
7253   new_state.sim, new_state.eim = nil, nil
7254   return head, new_state
7255 end
7256
7257 local function insert_numeric(head, state)
7258   local new
7259   local new_state = state
7260   if state.san and state.ean and state.san ~= state.ean then
7261     local d = node.new(DIR)
7262     d.dir = '+TLT'
7263     _, new = node.insert_before(head, state.san, d)
7264     if state.san == state.sim then state.sim = new end
7265     local d = node.new(DIR)
7266     d.dir = '-TLT'
7267     _, new = node.insert_after(head, state.ean, d)
7268     if state.ean == state.eim then state.eim = new end
7269   end
7270   new_state.san, new_state.ean = nil, nil
7271   return head, new_state
7272 end
7273
7274 -- TODO - \hbox with an explicit dir can lead to wrong results
7275 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7276 -- was s made to improve the situation, but the problem is the 3-dir
```

```

7277 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7278 -- well.
7279
7280 function Babel.bidi(head, ispar, hdir)
7281   local d -- d is used mainly for computations in a loop
7282   local prev_d = ''
7283   local new_d = false
7284
7285   local nodes = {}
7286   local outer_first = nil
7287   local inmath = false
7288
7289   local glue_d = nil
7290   local glue_i = nil
7291
7292   local has_en = false
7293   local first_et = nil
7294
7295   local has_hyperlink = false
7296
7297   local ATDIR = Babel.attr_dir
7298
7299   local save_outer
7300   local temp = node.get_attribute(head, ATDIR)
7301   if temp then
7302     temp = temp & 0x3
7303     save_outer = (temp == 0 and 'l') or
7304                  (temp == 1 and 'r') or
7305                  (temp == 2 and 'al')
7306   elseif ispar then -- Or error? Shouldn't happen
7307     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7308   else -- Or error? Shouldn't happen
7309     save_outer = ('TRT' == hdir) and 'r' or 'l'
7310   end
7311   -- when the callback is called, we are just _after_ the box,
7312   -- and the textdir is that of the surrounding text
7313   -- if not ispar and hdir ~= tex.textdir then
7314   --   save_outer = ('TRT' == hdir) and 'r' or 'l'
7315   -- end
7316   local outer = save_outer
7317   local last = outer
7318   -- 'al' is only taken into account in the first, current loop
7319   if save_outer == 'al' then save_outer = 'r' end
7320
7321   local fontmap = Babel.fontmap
7322
7323   for item in node.traverse(head) do
7324
7325     -- In what follows, #node is the last (previous) node, because the
7326     -- current one is not added until we start processing the neutrals.
7327
7328     -- three cases: glyph, dir, otherwise
7329     if item.id == GLYPH
7330        or (item.id == 7 and item.subtype == 2) then
7331
7332       local d_font = nil
7333       local item_r
7334       if item.id == 7 and item.subtype == 2 then
7335         item_r = item.replace -- automatic discs have just 1 glyph
7336       else
7337         item_r = item
7338       end
7339       local chardata = characters[item_r.char]

```

```

7340     d = chardata and chardata.d or nil
7341     if not d or d == 'nsm' then
7342         for nn, et in ipairs(ranges) do
7343             if item_r.char < et[1] then
7344                 break
7345             elseif item_r.char <= et[2] then
7346                 if not d then d = et[3]
7347                     elseif d == 'nsm' then d_font = et[3]
7348                 end
7349                 break
7350             end
7351         end
7352     end
7353     d = d or 'l'
7354
7355     -- A short 'pause' in bidi for mapfont
7356     d_font = d_font or d
7357     d_font = (d_font == 'l' and 0) or
7358             (d_font == 'nsm' and 0) or
7359             (d_font == 'r' and 1) or
7360             (d_font == 'al' and 2) or
7361             (d_font == 'an' and 2) or nil
7362     if d_font and fontmap and fontmap[d_font][item_r.font] then
7363         item_r.font = fontmap[d_font][item_r.font]
7364     end
7365
7366     if new_d then
7367         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7368         if inmath then
7369             attr_d = 0
7370         else
7371             attr_d = node.get_attribute(item, ATDIR)
7372             attr_d = attr_d & 0x3
7373         end
7374         if attr_d == 1 then
7375             outer_first = 'r'
7376             last = 'r'
7377         elseif attr_d == 2 then
7378             outer_first = 'r'
7379             last = 'al'
7380         else
7381             outer_first = 'l'
7382             last = 'l'
7383         end
7384         outer = last
7385         has_en = false
7386         first_et = nil
7387         new_d = false
7388     end
7389
7390     if glue_d then
7391         if (d == 'l' and 'l' or 'r') ~= glue_d then
7392             table.insert(nodes, {glue_i, 'on', nil})
7393         end
7394         glue_d = nil
7395         glue_i = nil
7396     end
7397
7398     elseif item.id == DIR then
7399         d = nil
7400
7401         if head ~= item then new_d = true end
7402

```

```

7403     elseif item.id == node.id'glue' and item.subtype == 13 then
7404         glue_d = d
7405         glue_i = item
7406         d = nil
7407
7408     elseif item.id == node.id'math' then
7409         inmath = (item.subtype == 0)
7410
7411     elseif item.id == 8 and item.subtype == 19 then
7412         has_hyperlink = true
7413
7414     else
7415         d = nil
7416     end
7417
7418     -- AL <= EN/ET/ES      -- W2 + W3 + W6
7419     if last == 'al' and d == 'en' then
7420         d = 'an'          -- W3
7421     elseif last == 'al' and (d == 'et' or d == 'es') then
7422         d = 'on'          -- W6
7423     end
7424
7425     -- EN + CS/ES + EN      -- W4
7426     if d == 'en' and #nodes >= 2 then
7427         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7428             and nodes[#nodes-1][2] == 'en' then
7429             nodes[#nodes][2] = 'en'
7430         end
7431     end
7432
7433     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
7434     if d == 'an' and #nodes >= 2 then
7435         if (nodes[#nodes][2] == 'cs')
7436             and nodes[#nodes-1][2] == 'an' then
7437             nodes[#nodes][2] = 'an'
7438         end
7439     end
7440
7441     -- ET/EN                  -- W5 + W7->l / W6->on
7442     if d == 'et' then
7443         first_et = first_et or (#nodes + 1)
7444     elseif d == 'en' then
7445         has_en = true
7446         first_et = first_et or (#nodes + 1)
7447     elseif first_et then      -- d may be nil here !
7448         if has_en then
7449             if last == 'l' then
7450                 temp = 'l'    -- W7
7451             else
7452                 temp = 'en'   -- W5
7453             end
7454         else
7455             temp = 'on'       -- W6
7456         end
7457         for e = first_et, #nodes do
7458             if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7459         end
7460         first_et = nil
7461         has_en = false
7462     end
7463
7464     -- Force mathdir in math if ON (currently works as expected only
7465     -- with 'l')

```

```

7466     if inmath and d == 'on' then
7467         d = ('TRT' == tex.mathdir) and 'r' or 'l'
7468     end
7469
7470     if d then
7471         if d == 'al' then
7472             d = 'r'
7473             last = 'al'
7474         elseif d == 'l' or d == 'r' then
7475             last = d
7476         end
7477         prev_d = d
7478         table.insert(nodes, {item, d, outer_first})
7479     end
7480
7481     outer_first = nil
7482
7483 end
7484
7485 -- TODO -- repeated here in case EN/ET is the last node. Find a
7486 -- better way of doing things:
7487 if first_et then      -- dir may be nil here !
7488     if has_en then
7489         if last == 'l' then
7490             temp = 'l'      -- W7
7491         else
7492             temp = 'en'     -- W5
7493         end
7494     else
7495         temp = 'on'        -- W6
7496     end
7497     for e = first_et, #nodes do
7498         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7499     end
7500 end
7501
7502 -- dummy node, to close things
7503 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7504
7505 ----- NEUTRAL -----
7506
7507 outer = save_outer
7508 last = outer
7509
7510 local first_on = nil
7511
7512 for q = 1, #nodes do
7513     local item
7514
7515     local outer_first = nodes[q][3]
7516     outer = outer_first or outer
7517     last = outer_first or last
7518
7519     local d = nodes[q][2]
7520     if d == 'an' or d == 'en' then d = 'r' end
7521     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7522
7523     if d == 'on' then
7524         first_on = first_on or q
7525     elseif first_on then
7526         if last == d then
7527             temp = d
7528         else

```

```

7529     temp = outer
7530 end
7531 for r = first_on, q - 1 do
7532     nodes[r][2] = temp
7533     item = nodes[r][1] -- MIRRORING
7534     if Babel.mirroring_enabled and item.id == GLYPH
7535         and temp == 'r' and characters[item.char] then
7536         local font_mode = ''
7537         if item.font > 0 and font.fonts[item.font].properties then
7538             font_mode = font.fonts[item.font].properties.mode
7539         end
7540         if font_mode ~= 'harf' and font_mode ~= 'plug' then
7541             item.char = characters[item.char].m or item.char
7542         end
7543     end
7544 end
7545 first_on = nil
7546 end
7547
7548 if d == 'r' or d == 'l' then last = d end
7549 end
7550
7551 ----- IMPLICIT, REORDER -----
7552
7553 outer = save_outer
7554 last = outer
7555
7556 local state = {}
7557 state.has_r = false
7558
7559 for q = 1, #nodes do
7560
7561     local item = nodes[q][1]
7562
7563     outer = nodes[q][3] or outer
7564
7565     local d = nodes[q][2]
7566
7567     if d == 'nsm' then d = last end -- W1
7568     if d == 'en' then d = 'an' end
7569     local isdir = (d == 'r' or d == 'l')
7570
7571     if outer == 'l' and d == 'an' then
7572         state.san = state.san or item
7573         state.ean = item
7574     elseif state.san then
7575         head, state = insert_numeric(head, state)
7576     end
7577
7578     if outer == 'l' then
7579         if d == 'an' or d == 'r' then -- im -> implicit
7580             if d == 'r' then state.has_r = true end
7581             state.sim = state.sim or item
7582             state.eim = item
7583         elseif d == 'l' and state.sim and state.has_r then
7584             head, state = insert_implicit(head, state, outer)
7585         elseif d == 'l' then
7586             state.sim, state.eim, state.has_r = nil, nil, false
7587         end
7588     else
7589         if d == 'an' or d == 'l' then
7590             if nodes[q][3] then -- nil except after an explicit dir
7591                 state.sim = item -- so we move sim 'inside' the group

```

```

7592         else
7593             state.sim = state.sim or item
7594         end
7595         state.eim = item
7596     elseif d == 'r' and state.sim then
7597         head, state = insert_implicit(head, state, outer)
7598     elseif d == 'r' then
7599         state.sim, state.eim = nil, nil
7600     end
7601 end
7602
7603 if isdir then
7604     last = d           -- Don't search back - best save now
7605 elseif d == 'on' and state.san then
7606     state.san = state.san or item
7607     state.ean = item
7608 end
7609
7610 end
7611
7612 head = node.prev(head) or head
7613
7614 ----- FIX HYPERLINKS -----
7615
7616 if has_hyperlink then
7617     local flag, linking = 0, 0
7618     for item in node.traverse(head) do
7619         if item.id == DIR then
7620             if item.dir == '+TRT' or item.dir == '+TLT' then
7621                 flag = flag + 1
7622             elseif item.dir == '-TRT' or item.dir == '-TLT' then
7623                 flag = flag - 1
7624             end
7625             elseif item.id == 8 and item.subtype == 19 then
7626                 linking = flag
7627             elseif item.id == 8 and item.subtype == 20 then
7628                 if linking > 0 then
7629                     if item.prev.id == DIR and
7630                        (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
7631                         d = node.new(DIR)
7632                         d.dir = item.prev.dir
7633                         node.remove(head, item.prev)
7634                         node.insert_after(head, item, d)
7635                     end
7636                 end
7637                 linking = 0
7638             end
7639         end
7640     end
7641
7642     return head
7643 end
7644 </basic>

```

## 10 Data for CJK

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

```

[0x0021]={c='ex'},
[0x0024]={c='pr'},
[0x0025]={c='po'},

```



```
[0x0028]={c='op'},
[0x0029]={c='cp'},
[0x002B]={c='pr'},
```

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

## 11 The ‘nil’ language

This ‘language’ does nothing, except setting the hyphenation patterns to nohyphenation. For this language currently no special definitions are needed or available. The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
7645 ⟨*nil⟩
7646 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ v⟨⟨version⟩⟩ Nil language]
7647 \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.

```
7648 \ifx\l@nil\undefined
7649   \newlanguage\l@nil
7650   \@namedef{bbl@hyphendata@the\l@nil}{}}}% Remove warning
7651   \let\bbl@elt\relax
7652   \edef\bbl@languages{% Add it to the list of languages
7653     \bbl@languages\bbl@elt{nil}{the\l@nil}}}%
7654 \fi
```

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

```
7655 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
\datenil
7656 \let\captionnil\@empty
7657 \let\datenil\@empty
```

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

```
7658 \def\bbl@inidata@nil{%
7659   \bbl@elt{identification}{tag.ini}{und}%
7660   \bbl@elt{identification}{load.level}{0}%
7661   \bbl@elt{identification}{charset}{utf8}%
7662   \bbl@elt{identification}{version}{1.0}%
7663   \bbl@elt{identification}{date}{2022-05-16}%
7664   \bbl@elt{identification}{name.local}{nil}%
7665   \bbl@elt{identification}{name.english}{nil}%
7666   \bbl@elt{identification}{name.babel}{nil}%
7667   \bbl@elt{identification}{tag.bcp47}{und}%
7668   \bbl@elt{identification}{language.tag.bcp47}{und}%
7669   \bbl@elt{identification}{tag.opentype}{dflt}%
7670   \bbl@elt{identification}{script.name}{Latin}%
7671   \bbl@elt{identification}{script.tag.bcp47}{Latn}%
7672   \bbl@elt{identification}{script.tag.opentype}{DFLT}%
7673   \bbl@elt{identification}{level}{1}%
7674   \bbl@elt{identification}{encodings}}}%
7675   \bbl@elt{identification}{derivate}{no}}
7676 \@namedef{bbl@tbc@nil}{und}
7677 \@namedef{bbl@lbc@nil}{und}
7678 \@namedef{bbl@casing@nil}{und} % TODO
7679 \@namedef{bbl@lotf@nil}{dflt}
7680 \@namedef{bbl@elname@nil}{nil}
7681 \@namedef{bbl@lname@nil}{nil}
7682 \@namedef{bbl@esname@nil}{Latin}
```

```

7683 \@namedef{bbl@sname@nil}{Latin}
7684 \@namedef{bbl@sbc@nil}{Latn}
7685 \@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.

```

7686 \ldf@finish{nil}
7687 \</nil>

```

## 12 Calendars

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

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

```

7688 <<Compute Julian day>> ≡
7689 \def\bbl@fpmo#1#2{(#1-#2*floo(#1/#2))}
7690 \def\bbl@cs@gregleap#1{%
7691   (\bbl@fpmo{#1}{4} == 0) &&
7692   (!((\bbl@fpmo{#1}{100} == 0) && (\bbl@fpmo{#1}{400} != 0)))}
7693 \def\bbl@cs@jd#1#2#3{ % year, month, day
7694   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
7695     floo((#1 - 1) / 4) + (-floo((#1 - 1) / 100)) +
7696     floo((#1 - 1) / 400) + floo((((367 * #2) - 362) / 12) +
7697     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3) }}
7698 <</Compute Julian day>>

```

### 12.1 Islamic

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

```

7699 <ca-islamic>
7700 \ExplSyntaxOn
7701 <<Compute Julian day>>
7702 % == islamic (default)
7703 % Not yet implemented
7704 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{

```

The Civil calendar:

```

7705 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
7706   ((#3 + ceil(29.5 * (#2 - 1)) +
7707     (#1 - 1) * 354 + floo((3 + (11 * #1)) / 30) +
7708     1948439.5) - 1) }
7709 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
7710 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
7711 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
7712 \def\bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
7713 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
7714 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
7715   \edef\bbl@tempa{%
7716     \fp_eval:n{ floo(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
7717   \edef#5{%
7718     \fp_eval:n{ floo(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
7719   \edef#6{\fp_eval:n{
7720     min(12, ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
7721   \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  $\sim 1435/\sim 1460$  (Gregorian  $\sim 2014/\sim 2038$ ).

```

7722 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
7723   56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%

```

```

7724 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
7725 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
7726 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
7727 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
7728 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
7729 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
7730 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
7731 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
7732 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
7733 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
7734 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
7735 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
7736 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
7737 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
7738 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
7739 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
7740 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
7741 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
7742 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
7743 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
7744 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
7745 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
7746 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
7747 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
7748 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
7749 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
7750 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
7751 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
7752 65401,65431,65460,65490,65520}
7753 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
7754 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
7755 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
7756 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
7757   \ifnum#2>2014 \ifnum#2<2038
7758     \bbl@afterfi\expandafter\@gobble
7759   \fi\fi
7760   {\bbl@error{Year-out-of-range}{The-allowed-range-is-2014-2038}}%
7761   \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
7762     \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
7763   \count@\@ne
7764   \bbl@foreach\bbl@cs@umalqura@data{%
7765     \advance\count@\@ne
7766     \ifnum##1>\bbl@tempd\else
7767       \edef\bbl@tempe{\the\count@}%
7768       \edef\bbl@tempb{##1}%
7769     \fi}%
7770   \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
7771   \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1 ) / 12) }}% annus
7772   \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
7773   \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
7774   \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
7775 \ExplSyntaxOff
7776 \bbl@add\bbl@precalendar{%
7777   \bbl@replace\bbl@ld@calendar{-civil}{}%
7778   \bbl@replace\bbl@ld@calendar{-umalqura}{}%
7779   \bbl@replace\bbl@ld@calendar{+}{}%
7780   \bbl@replace\bbl@ld@calendar{-}{}}
7781 /ca-islamic)

```

## 12.2 Hebrew

This is basically the set of macros written by Michail Rozman in 1991, with corrections and adaptations by Rama Porrat, Misha, Dan Haran and Boris Lavva. This must be eventually replaced by

computations with l3fp. An explanation of what's going on can be found in hebcal.sty

```

7782 (*ca-hebrew)
7783 \newcount\bbl@cntcommon
7784 \def\bbl@remainder#1#2#3{%
7785   #3=#1\relax
7786   \divide #3 by #2\relax
7787   \multiply #3 by -#2\relax
7788   \advance #3 by #1\relax}%
7789 \newif\ifbbl@divisible
7790 \def\bbl@checkifdivisible#1#2{%
7791   {\countdef\tmp=0
7792   \bbl@remainder{#1}{#2}{\tmp}%
7793   \ifnum \tmp=0
7794     \global\bbl@divisibletrue
7795   \else
7796     \global\bbl@divisiblefalse
7797   \fi}}
7798 \newif\ifbbl@gregleap
7799 \def\bbl@ifgregleap#1{%
7800   \bbl@checkifdivisible{#1}{4}%
7801   \ifbbl@divisible
7802     \bbl@checkifdivisible{#1}{100}%
7803     \ifbbl@divisible
7804       \bbl@checkifdivisible{#1}{400}%
7805       \ifbbl@divisible
7806         \bbl@gregleaptrue
7807       \else
7808         \bbl@gregleapfalse
7809       \fi
7810     \else
7811       \bbl@gregleaptrue
7812     \fi
7813   \else
7814     \bbl@gregleapfalse
7815   \fi
7816   \ifbbl@gregleap}
7817 \def\bbl@gregdayspriormonths#1#2#3{%
7818   {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
7819     181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
7820   \bbl@ifgregleap{#2}%
7821   \ifnum #1 > 2
7822     \advance #3 by 1
7823   \fi
7824   \fi
7825   \global\bbl@cntcommon=#3}%
7826   #3=\bbl@cntcommon}
7827 \def\bbl@gregdaysprioryears#1#2{%
7828   {\countdef\tmpc=4
7829   \countdef\tmpb=2
7830   \tmpb=#1\relax
7831   \advance \tmpb by -1
7832   \tmpc=\tmpb
7833   \multiply \tmpc by 365
7834   #2=\tmpc
7835   \tmpc=\tmpb
7836   \divide \tmpc by 4
7837   \advance #2 by \tmpc
7838   \tmpc=\tmpb
7839   \divide \tmpc by 100
7840   \advance #2 by -\tmpc
7841   \tmpc=\tmpb
7842   \divide \tmpc by 400
7843   \advance #2 by \tmpc

```

```

7844 \global\bbl@cntcommon=#2\relax}%
7845 #2=\bbl@cntcommon}
7846 \def\bbl@absfromgreg#1#2#3#4{%
7847 {\countdef\tmpd=0
7848 #4=#1\relax
7849 \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
7850 \advance #4 by \tmpd
7851 \bbl@gregdaysprioryears{#3}{\tmpd}%
7852 \advance #4 by \tmpd
7853 \global\bbl@cntcommon=#4\relax}%
7854 #4=\bbl@cntcommon}
7855 \newif\ifbbl@hebrleap
7856 \def\bbl@checkleaphebryear#1{%
7857 {\countdef\tmpa=0
7858 \countdef\tmpb=1
7859 \tmpa=#1\relax
7860 \multiply \tmpa by 7
7861 \advance \tmpa by 1
7862 \bbl@remainder{\tmpa}{19}{\tmpb}%
7863 \ifnum \tmpb < 7
7864 \global\bbl@hebrleaptrue
7865 \else
7866 \global\bbl@hebrleapfalse
7867 \fi}}
7868 \def\bbl@hebrlapsedmonths#1#2{%
7869 {\countdef\tmpa=0
7870 \countdef\tmpb=1
7871 \countdef\tmpc=2
7872 \tmpa=#1\relax
7873 \advance \tmpa by -1
7874 #2=\tmpa
7875 \divide #2 by 19
7876 \multiply #2 by 235
7877 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
7878 \tmpc=\tmpb
7879 \multiply \tmpb by 12
7880 \advance #2 by \tmpb
7881 \multiply \tmpc by 7
7882 \advance \tmpc by 1
7883 \divide \tmpc by 19
7884 \advance #2 by \tmpc
7885 \global\bbl@cntcommon=#2}%
7886 #2=\bbl@cntcommon}
7887 \def\bbl@hebrlapseddays#1#2{%
7888 {\countdef\tmpa=0
7889 \countdef\tmpb=1
7890 \countdef\tmpc=2
7891 \bbl@hebrlapsedmonths{#1}{#2}%
7892 \tmpa=#2\relax
7893 \multiply \tmpa by 13753
7894 \advance \tmpa by 5604
7895 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
7896 \divide \tmpa by 25920
7897 \multiply #2 by 29
7898 \advance #2 by 1
7899 \advance #2 by \tmpa
7900 \bbl@remainder{#2}{7}{\tmpa}%
7901 \ifnum \tmpc < 19440
7902 \ifnum \tmpc < 9924
7903 \else
7904 \ifnum \tmpa=2
7905 \bbl@checkleaphebryear{#1}% of a common year
7906 \ifbbl@hebrleap

```

```

7907             \else
7908                 \advance #2 by 1
7909             \fi
7910         \fi
7911     \fi
7912     \ifnum \tmpc < 16789
7913     \else
7914         \ifnum \tmpa=1
7915             \advance #1 by -1
7916             \bbl@checkleaphebrewyear{#1}% at the end of leap year
7917             \ifbbl@hebrleap
7918                 \advance #2 by 1
7919             \fi
7920         \fi
7921     \fi
7922 \else
7923     \advance #2 by 1
7924 \fi
7925 \bbl@remainder{#2}{7}{\tmpa}%
7926 \ifnum \tmpa=0
7927     \advance #2 by 1
7928 \else
7929     \ifnum \tmpa=3
7930         \advance #2 by 1
7931     \else
7932         \ifnum \tmpa=5
7933             \advance #2 by 1
7934         \fi
7935     \fi
7936 \fi
7937 \global\bbl@cntcommon=#2\relax}%
7938 #2=\bbl@cntcommon}
7939 \def\bbl@daysinhebrewyear#1#2{%
7940 {\countdef\tmpe=12
7941  \bbl@hebreleapseddays{#1}{\tmpe}%
7942  \advance #1 by 1
7943  \bbl@hebreleapseddays{#1}{#2}%
7944  \advance #2 by -\tmpe
7945  \global\bbl@cntcommon=#2}%
7946  #2=\bbl@cntcommon}
7947 \def\bbl@hebrdayspriormonths#1#2#3{%
7948 {\countdef\tmpf= 14
7949  #3=\ifcase #1\relax
7950      0 \or
7951      0 \or
7952      30 \or
7953      59 \or
7954      89 \or
7955      118 \or
7956      148 \or
7957      148 \or
7958      177 \or
7959      207 \or
7960      236 \or
7961      266 \or
7962      295 \or
7963      325 \or
7964      400
7965  \fi
7966  \bbl@checkleaphebrewyear{#2}%
7967  \ifbbl@hebrleap
7968      \ifnum #1 > 6
7969          \advance #3 by 30

```

```

7970     \fi
7971 \fi
7972 \bbl@daysinhebrewyear{#2}{\tmpf}%
7973 \ifnum #1 > 3
7974     \ifnum \tmpf=353
7975         \advance #3 by -1
7976     \fi
7977     \ifnum \tmpf=383
7978         \advance #3 by -1
7979     \fi
7980 \fi
7981 \ifnum #1 > 2
7982     \ifnum \tmpf=355
7983         \advance #3 by 1
7984     \fi
7985     \ifnum \tmpf=385
7986         \advance #3 by 1
7987     \fi
7988 \fi
7989 \global\bbl@cntcommon=#3\relax}%
7990 #3=\bbl@cntcommon}
7991 \def\bbl@absfromhebr#1#2#3#4{%
7992     {#4=#1\relax
7993     \bbl@hebrdayspriormonths{#2}{#3}{#1}%
7994     \advance #4 by #1\relax
7995     \bbl@hebrrelapseddays{#3}{#1}%
7996     \advance #4 by #1\relax
7997     \advance #4 by -1373429
7998     \global\bbl@cntcommon=#4\relax}%
7999 #4=\bbl@cntcommon}
8000 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8001     {\countdef\tmpx= 17
8002     \countdef\tmpy= 18
8003     \countdef\tmpz= 19
8004     #6=#3\relax
8005     \global\advance #6 by 3761
8006     \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8007     \tmpz=1 \tmpy=1
8008     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8009     \ifnum \tmpx > #4\relax
8010         \global\advance #6 by -1
8011         \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8012     \fi
8013     \advance #4 by -\tmpx
8014     \advance #4 by 1
8015     #5=#4\relax
8016     \divide #5 by 30
8017     \loop
8018         \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8019         \ifnum \tmpx < #4\relax
8020             \advance #5 by 1
8021             \tmpy=\tmpx
8022         \repeat
8023     \global\advance #5 by -1
8024     \global\advance #4 by -\tmpy}}
8025 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebrewyear
8026 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8027 \def\bbl@ca@hebrew#1-#2-#3\@#4#5#6{%
8028     \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8029     \bbl@hebrfromgreg
8030     {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8031     {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebrewyear}%
8032     \edef#4{\the\bbl@hebrewyear}%

```

```

8033 \edef#5{\the\bbl@hebrmonth}%
8034 \edef#6{\the\bbl@hebrday}}
8035 \</ca-hebrew>

```

## 12.3 Persian

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

```

8036 \<ca-persian>
8037 \ExplSyntaxOn
8038 \<<Compute Julian day>>
8039 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8040 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8041 \def\bbl@ca@persian#1-#2-#3\@#4#5#6{%
8042 \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8043 \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8044 \bbl@afterfi\expandafter\@gobble
8045 \fi\fi
8046 {\bbl@error{Year-out-of-range}{The-allowed-range-is-2013-2050}}%
8047 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8048 \ifin\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8049 \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8050 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8051 \ifnum\bbl@tempc<\bbl@tempb
8052 \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8053 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8054 \ifin\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8055 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8056 \fi
8057 \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8058 \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8059 \edef#5{\fp_eval:n{% set Jalali month
8060 (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8061 \edef#6{\fp_eval:n{% set Jalali day
8062 (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6))}}}}
8063 \ExplSyntaxOff
8064 \</ca-persian>

```

## 12.4 Coptic and Ethiopic

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

```

8065 \<ca-coptic>
8066 \ExplSyntaxOn
8067 \<<Compute Julian day>>
8068 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
8069 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8070 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8071 \edef#4{\fp_eval:n{%
8072 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8073 \edef\bbl@tempc{\fp_eval:n{%
8074 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8075 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8076 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}%
8077 \ExplSyntaxOff
8078 \</ca-coptic>
8079 \<ca-ethiopic>
8080 \ExplSyntaxOn
8081 \<<Compute Julian day>>

```



## 12.5 Buddhist

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

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

```
8114 \bplain\la plain.tex
8115 \bplain\la 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.

```
8116 \bplain\def\fmtname{babel-plain}
8117 \bplain\def\fmtname{babel-lplain}
```

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

## 13.2 Emulating some $\text{\LaTeX}$ 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).

```
8118 \langle\langle *Emulate LaTeX \rangle\rangle \equiv
8119 \def\@empty{}
8120 \def\loadlocalcfg#1{%
8121   \openin0#1.cfg
8122   \ifeof0
8123     \closein0
8124   \else
8125     \closein0
8126     {\immediate\write16{*****}%
8127      \immediate\write16{* Local config file #1.cfg used}%
8128      \immediate\write16{*}%
8129     }
8130   \input #1.cfg\relax
8131 \fi
8132 \@endoflfd}
```

## 13.3 General tools

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

```
8133 \long\def\@firstofone#1{#1}
8134 \long\def\@firstoftwo#1#2{#1}
8135 \long\def\@secondoftwo#1#2{#2}
8136 \def\@nnil{\@nil}
8137 \def\@gobbletwo#1#2{}
8138 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
8139 \def\@star@or@long#1{%
8140   \@ifstar
8141   {\let\l@ngrel@x\relax#1}%
8142   {\let\l@ngrel@x\long#1}}
8143 \let\l@ngrel@x\relax
8144 \def\@car#1#2\@nil{#1}
8145 \def\@cdr#1#2\@nil{#2}
8146 \let\@typeset@protect\relax
8147 \let\protected@edef\edef
8148 \long\def\@gobble#1{}
8149 \edef\@backslashchar{\expandafter\@gobble\string\}
8150 \def\strip@prefix#1>{}
8151 \def\g@addto@macro#1#2{{%
8152   \toks@{\expandafter{#1#2}}%
8153   \xdef#1{\the\toks@}}}
8154 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8155 \def\@nameuse#1{\csname #1\endcsname}
```

```

8156 \def\@ifundefined#1{%
8157   \expandafter\ifx\csname#1\endcsname\relax
8158     \expandafter\@firstoftwo
8159   \else
8160     \expandafter\@secondoftwo
8161   \fi}
8162 \def\@expandtwoargs#1#2#3{%
8163   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8164 \def\zap@space#1 #2{%
8165   #1%
8166   \ifx#2@empty\else\expandafter\zap@space\fi
8167   #2}
8168 \let\bbl@trace\@gobble
8169 \def\bbl@error#1#2{%
8170   \begingroup
8171     \newlinechar=`^^J
8172     \def\{^^J(babel) }%
8173     \errhelp{#2}\errmessage{\{#1}%
8174   \endgroup}
8175 \def\bbl@warning#1{%
8176   \begingroup
8177     \newlinechar=`^^J
8178     \def\{^^J(babel) }%
8179     \message{\{#1}%
8180   \endgroup}
8181 \let\bbl@infowarn\bbl@warning
8182 \def\bbl@info#1{%
8183   \begingroup
8184     \newlinechar=`^^J
8185     \def\{^^J}%
8186     \wlog{#1}%
8187   \endgroup}

```

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

```

8188 \ifx\@preamblecmds\@undefined
8189   \def\@preamblecmds{}
8190 \fi
8191 \def\@onlypreamble#1{%
8192   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8193     \@preamblecmds\do#1}}
8194 \@onlypreamble\@onlypreamble

```

Mimick  $\LaTeX$ 's `\AtBeginDocument`; for this to work the user needs to add `\begin{document}` to his file.

```

8195 \def\begin{document}{%
8196   \@begin{document}hook
8197   \global\let\@begin{document}hook\@undefined
8198   \def\do##1{\global\let##1\@undefined}%
8199   \@preamblecmds
8200   \global\let\do\noexpand}
8201 \ifx\@begin{document}hook\@undefined
8202   \def\@begin{document}hook{}
8203 \fi
8204 \@onlypreamble\@begin{document}hook
8205 \def\AtBeginDocument{\g@addto@macro\@begin{document}hook}

```

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

```

8206 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
8207 \@onlypreamble\AtEndOfPackage
8208 \def\@endoflfd{}
8209 \@onlypreamble\@endoflfd
8210 \let\bbl@afterlang\@empty
8211 \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.

```

8212 \catcode`\&=\z@
8213 \ifx&\if@filesw\undefined
8214   \expandafter\let\csname if@filesw\expandafter\endcsname
8215     \csname iffalse\endcsname
8216 \fi
8217 \catcode`\&=4

```

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

```

8218 \def\newcommand{\@star@or@long\newcommand}
8219 \def\newcommand#1{%
8220   \@testopt{\@newcommand#1}0}
8221 \def\@newcommand#1[#2]{%
8222   \@ifnextchar [{\@xargdef#1[#2]}%
8223     {\@argdef#1[#2]}}
8224 \long\def\@argdef#1[#2]#3{%
8225   \@yargdef#1\@ne{#2}{#3}}
8226 \long\def\@xargdef#1[#2][#3]#4{%
8227   \expandafter\def\expandafter#1\expandafter{%
8228     \expandafter\@protected@testopt\expandafter #1%
8229     \csname\string#1\expandafter\endcsname{#3}}%
8230   \expandafter\@yargdef \csname\string#1\endcsname
8231   \tw@{#2}{#4}}
8232 \long\def\@yargdef#1#2#3{%
8233   \@tempcnta#3\relax
8234   \advance \@tempcnta \@ne
8235   \let\@hash@\relax
8236   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
8237   \@tempcntb #2%
8238   \@whilenum\@tempcntb <\@tempcnta
8239   \do{%
8240     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
8241     \advance\@tempcntb \@ne}%
8242   \let\@hash@###
8243   \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
8244 \def\providecommand{\@star@or@long\providecommand}
8245 \def\providecommand#1{%
8246   \begingroup
8247     \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
8248   \endgroup
8249   \expandafter\@ifundefined\@gtempa
8250     {\def\reserved@a{\newcommand#1}}%
8251     {\let\reserved@a\relax
8252     \def\reserved@a{\newcommand\reserved@a}}%
8253   \reserved@a}%
8254 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
8255 \def\declare@robustcommand#1{%
8256   \edef\reserved@a{\string#1}%
8257   \def\reserved@b{#1}%
8258   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
8259   \edef#1{%
8260     \ifx\reserved@a\reserved@b
8261       \noexpand\x@protect
8262       \noexpand#1%
8263     \fi
8264     \noexpand\protect
8265     \expandafter\noexpand\csname
8266       \expandafter\@gobble\string#1 \endcsname
8267   }%
8268   \expandafter\newcommand\csname
8269     \expandafter\@gobble\string#1 \endcsname

```

```

8270 }
8271 \def\x@protect#1{%
8272   \ifx\protect\@typeset@protect\else
8273     \@x@protect#1%
8274   \fi
8275 }
8276 \catcode`\&=\z@ % Trick to hide conditionals
8277 \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`.

```

8278 \def\bbl@tempa{\csname newif\endcsname&fin@}
8279 \catcode`\&=4
8280 \ifx\in@\@undefined
8281   \def\in@#1#2{%
8282     \def\in@@##1#1##2##3\in@@{%
8283       \ifx\in@@##2\in@false\else\in@true\fi}%
8284     \in@@#2#1\in@\in@@}
8285 \else
8286   \let\bbl@tempa\@empty
8287 \fi
8288 \bbl@tempa

```

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

```

8289 \def\@ifpackagewith#1#2#3#4{#3}

```

The  $\text{\TeX}$  macro `\@ifl@aded` checks whether a file was loaded. This functionality is not needed for plain  $\text{\TeX}$  but we need the macro to be defined as a no-op.

```

8290 \def\@ifl@aded#1#2#3#4{}

```

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

```

8291 \ifx\@tempcnta\@undefined
8292   \csname newcount\endcsname\@tempcnta\relax
8293 \fi
8294 \ifx\@tempcntb\@undefined
8295   \csname newcount\endcsname\@tempcntb\relax
8296 \fi

```

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

```

8297 \ifx\bye\@undefined
8298   \advance\count10 by -2\relax
8299 \fi
8300 \ifx\@ifnextchar\@undefined
8301   \def\@ifnextchar#1#2#3{%
8302     \let\reserved@d=#1%
8303     \def\reserved@a{#2}\def\reserved@b{#3}%
8304     \futurelet\@let@token\@ifnch}
8305   \def\@ifnch{%
8306     \ifx\@let@token\@sptoken
8307       \let\reserved@c\@ifnch
8308     \else
8309       \ifx\@let@token\reserved@d
8310         \let\reserved@c\reserved@a
8311       \else
8312         \let\reserved@c\reserved@b
8313     \fi

```

```

8314 \fi
8315 \reserved@c}
8316 \def\:\let\sptoken= } \: % this makes \sptoken a space token
8317 \def\:\xifnch} \expandafter\def\:\futurelet\@let@token\@ifnch}
8318 \fi
8319 \def\@testopt#1#2{%
8320 \@ifnextchar[{\#1}{\#1[\#2]}}
8321 \def\@protected@testopt#1{%
8322 \ifx\protect\@typeset@protect
8323 \expandafter\@testopt
8324 \else
8325 \@x@protect#1%
8326 \fi}
8327 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8328 #2\relax}\fi}
8329 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8330 \else\expandafter\@gobble\fi{#1}}

```

### 13.4 Encoding related macros

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

```

8331 \def\DeclareTextCommand{%
8332 \@dec@text@cmd\providecommand
8333 }
8334 \def\ProvideTextCommand{%
8335 \@dec@text@cmd\providecommand
8336 }
8337 \def\DeclareTextSymbol#1#2#3{%
8338 \@dec@text@cmd\chardef#1{#2}#3\relax
8339 }
8340 \def\@dec@text@cmd#1#2#3{%
8341 \expandafter\def\expandafter#2%
8342 \expandafter{%
8343 \csname#3-cmd\expandafter\endcsname
8344 \expandafter#2%
8345 \csname#3\string#2\endcsname
8346 }%
8347 % \let\@ifdefinable\rc@ifdefinable
8348 \expandafter#1\csname#3\string#2\endcsname
8349 }
8350 \def\@current@cmd#1{%
8351 \ifx\protect\@typeset@protect\else
8352 \noexpand#1\expandafter\@gobble
8353 \fi
8354 }
8355 \def\@changed@cmd#1#2{%
8356 \ifx\protect\@typeset@protect
8357 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8358 \expandafter\ifx\csname ?\string#1\endcsname\relax
8359 \expandafter\def\csname ?\string#1\endcsname{%
8360 \@changed@x@err{#1}%
8361 }%
8362 \fi
8363 \global\expandafter\let
8364 \csname\cf@encoding\string#1\expandafter\endcsname
8365 \csname ?\string#1\endcsname
8366 \fi
8367 \csname\cf@encoding\string#1%
8368 \expandafter\endcsname
8369 \else
8370 \noexpand#1%
8371 \fi
8372 }

```

```

8373 \def\@changed@x@err#1{%
8374     \errhelp{Your command will be ignored, type <return> to proceed}%
8375     \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8376 \def\DeclareTextCommandDefault#1{%
8377     \DeclareTextCommand#1?%
8378 }
8379 \def\ProvideTextCommandDefault#1{%
8380     \ProvideTextCommand#1?%
8381 }
8382 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8383 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8384 \def\DeclareTextAccent#1#2#3{%
8385     \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
8386 }
8387 \def\DeclareTextCompositeCommand#1#2#3#4{%
8388     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8389     \edef\reserved@b{\string##1}%
8390     \edef\reserved@c{%
8391         \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8392     \ifx\reserved@b\reserved@c
8393         \expandafter\expandafter\expandafter\ifx
8394             \expandafter\@car\reserved@a\relax\relax\@nil
8395             \@text@composite
8396         \else
8397             \edef\reserved@b##1{%
8398                 \def\expandafter\noexpand
8399                     \csname#2\string#1\endcsname####1{%
8400                         \noexpand\@text@composite
8401                             \expandafter\noexpand\csname#2\string#1\endcsname
8402                             ####1\noexpand\@empty\noexpand\@text@composite
8403                             {##1}%
8404                         }%
8405                     }%
8406             \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8407         \fi
8408         \expandafter\def\csname\expandafter\string\csname
8409             #2\endcsname\string#1-\string#3\endcsname{#4}
8410     \else
8411         \errhelp{Your command will be ignored, type <return> to proceed}%
8412         \errmessage{\string\DeclareTextCompositeCommand\space used on
8413             inappropriate command \protect#1}
8414     \fi
8415 }
8416 \def\@text@composite#1#2#3\@text@composite{%
8417     \expandafter\@text@composite@x
8418         \csname\string#1-\string#2\endcsname
8419 }
8420 \def\@text@composite@x#1#2{%
8421     \ifx#1\relax
8422         #2%
8423     \else
8424         #1%
8425     \fi
8426 }
8427 %
8428 \def\@strip@args#1:#2-#3\@strip@args{#2}
8429 \def\DeclareTextComposite#1#2#3#4{%
8430     \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8431     \bgroup
8432         \lccode`\@=#4%
8433         \lowercase{%
8434             \egroup
8435             \reserved@a @%

```

```

8436 }%
8437 }
8438 %
8439 \def\UseTextSymbol#1#2{#2}
8440 \def\UseTextAccent#1#2#3{}
8441 \def\@use@text@encoding#1{}
8442 \def\DeclareTextSymbolDefault#1#2{%
8443   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8444 }
8445 \def\DeclareTextAccentDefault#1#2{%
8446   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8447 }
8448 \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.

```

8449 \DeclareTextAccent{"}{OT1}{127}
8450 \DeclareTextAccent{'}{OT1}{19}
8451 \DeclareTextAccent{^}{OT1}{94}
8452 \DeclareTextAccent{`}{OT1}{18}
8453 \DeclareTextAccent{~}{OT1}{126}

```

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

```

8454 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8455 \DeclareTextSymbol{\textquotedblright}{OT1}{\`"}
8456 \DeclareTextSymbol{\textquoteleft}{OT1}{\`'}
8457 \DeclareTextSymbol{\textquoteright}{OT1}{\`'}
8458 \DeclareTextSymbol{\i}{OT1}{16}
8459 \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`.

```

8460 \ifx\scriptsize\@undefined
8461   \let\scriptsize\sevenrm
8462 \fi

```

And a few more “dummy” definitions.

```

8463 \def\language{english}%
8464 \let\bbl@opt@shorthands\@nnil
8465 \def\bbl@ifshorthand#1#2#3{#2}%
8466 \let\bbl@language@opts\@empty
8467 \let\bbl@ensureinfo\@gobble
8468 \let\bbl@provide@locale\relax
8469 \ifx\babeloptionstrings\@undefined
8470   \let\bbl@opt@strings\@nnil
8471 \else
8472   \let\bbl@opt@strings\babeloptionstrings
8473 \fi
8474 \def\BabelStringsDefault{generic}
8475 \def\bbl@tempa{normal}
8476 \ifx\babeloptionmath\bbl@tempa
8477   \def\bbl@mathnormal{\noexpand\textormath}
8478 \fi
8479 \def\AfterBabelLanguage#1#2{}
8480 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
8481 \let\bbl@afterlang\relax
8482 \def\bbl@opt@safe{BR}
8483 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
8484 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
8485 \expandafter\newif\csname ifbbl@single\endcsname
8486 \chardef\bbl@bidimode\z@
8487 <</Emulate LaTeX>>

```

A proxy file:



```

8488 <*plain>
8489 \input babel.def
8490 </plain>

```

## 14 Acknowledgements

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

## References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national  $\text{\LaTeX}$  styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The  $\text{\TeX}$ book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport,  *$\text{\LaTeX}$ , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in:  $\text{\TeX}$ 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  $\text{\TeX}$* , *TUGboat* 9 (1988) #1, p. 70–72.
- [11] Joachim Schrod, *International  $\text{\LaTeX}$  is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using  $\text{\LaTeX}$* , Springer, 2002, p. 301–373.
- [13] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).