

Babel

Code

Version 3.94
2023/09/12

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Localization and
internationalization

Unicode

TEX

pdfTEX

LuaTEX

XeTEX

Contents

1	Identification and loading of required files	3
2	locale directory	3
3	Tools	3
3.1	Multiple languages	7
3.2	The Package File (<code>\LaTeX</code> , <code>babel.sty</code>)	8
3.3	<code>base</code>	9
3.4	<code>key=value</code> options and other general option	10
3.5	Conditional loading of shorthands	11
3.6	Interlude for Plain	13
4	Multiple languages	13
4.1	Selecting the language	15
4.2	Errors	23
4.3	Hooks	25
4.4	Setting up language files	27
4.5	Shorthands	29
4.6	Language attributes	38
4.7	Support for saving macro definitions	40
4.8	Short tags	41
4.9	Hyphens	42
4.10	Multiencoding strings	43
4.11	Macros common to a number of languages	49
4.12	Making glyphs available	49
4.12.1	Quotation marks	50
4.12.2	Letters	51
4.12.3	Shorthands for quotation marks	52
4.12.4	Umlauts and tremas	53
4.13	Layout	54
4.14	Load engine specific macros	54
4.15	Creating and modifying languages	55
5	Adjusting the Babel behavior	77
5.1	Cross referencing macros	79
5.2	Marks	82
5.3	Preventing clashes with other packages	83
5.3.1	<code>ifthen</code>	83
5.3.2	<code>varioref</code>	83
5.3.3	<code>hhline</code>	84
5.4	Encoding and fonts	84
5.5	Basic bidi support	86
5.6	Local Language Configuration	89
5.7	Language options	90
6	The kernel of Babel (<code>babel.def</code>, <code>common</code>)	93
7	Loading hyphenation patterns	93
8	Font handling with <code>fontspec</code>	97
9	Hooks for XeTeX and LuaTeX	101
9.1	XeTeX	101
9.2	Layout	103
9.3	8-bit TeX	104
9.4	LuaTeX	105
9.5	Southeast Asian scripts	111
9.6	CJK line breaking	113

9.7	Arabic justification	115
9.8	Common stuff	119
9.9	Automatic fonts and ids switching	119
9.10	Bidi	125
9.11	Layout	127
9.12	Lua: transforms	135
9.13	Lua: Auto bidi with <code>basic</code> and <code>basic-r</code>	143
10	Data for CJK	154
11	The ‘nil’ language	154
12	Calendars	155
12.1	Islamic	155
12.2	Hebrew	157
12.3	Persian	161
12.4	Coptic and Ethiopic	162
12.5	Buddhist	162
13	Support for Plain \TeX (<code>plain.def</code>)	163
13.1	Not renaming <code>hyphen.tex</code>	163
13.2	Emulating some \LaTeX features	164
13.3	General tools	165
13.4	Encoding related macros	168
14	Acknowledgements	171

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

1 Identification and loading of required files

Code documentation is still under revision.

The babel package after unpacking consists of the following files:

babel.sty is the L^AT_EX package, which set options and load language styles.

babel.def is loaded by Plain.

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

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

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

There some additional tex, def and lua files

The babel installer extends docstrip with a few “pseudo-guards” to set “variables” used at installation time. They are used with <@name> at the 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=3.94>>
2 <<date=2023/09/12>>
```

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

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

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

```

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

```

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

```

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

```

`\bbl@afterelse` Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the `\else` and `\fi` parts of an `\if`-statement¹. These macros will break if another `\if... \fi` statement appears in one of the arguments and it is not enclosed in braces.

```

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

```

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

```

33 \def\bbl@exp#1{%
34   \begingroup
35   \let\<\noexpand
36   \let\<\bbl@exp@en
37   \let\[\bbl@exp@ue
38   \edef\bbl@exp@aux{\endgroup#1}%
39   \bbl@exp@aux}
40 \def\bbl@exp@en#1>{\expandafter\noexpand\c#1\endcsname}%
41 \def\bbl@exp@ue#1]{%
42   \unexpanded\expandafter\expandafter\expandafter{\c#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 \@nil{\bbl@trim@i##1}}
53 \bbl@tempa{ }
54 \long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
55 \long\def\bbl@trim@def#1{\bbl@trim{def#1}}

```

`\bbl@ifunset` To check if a macro is defined, we create a new macro, which does the same as `\@ifundefined`. However, in an ϵ -tex engine, it is based on `\ifcsname`, which is more efficient, and does not waste

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

101 \def\bb@replace#1#2#3{% in #1 -> repl #2 by #3
102   \toks@{}}
103 \def\bb@replace@aux##1#2##2#2{%

```

```

104 \ifx\bbbl@nil##2%
105   \toks@\expandafter{\the\toks@##1}%
106 \else
107   \toks@\expandafter{\the\toks@##1#3}%
108   \bbbl@afterfi
109   \bbbl@replace@aux##2#2%
110   \fi}%
111 \expandafter\bbbl@replace@aux#1#2\bbbl@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 \bbbl@TG@@date, and also fails if there are macros with spaces, because they are retokenized). It may change! (or even merged with \bbbl@replace; I'm not sure cchecking the replacement is really necessary or just paranoia).

```

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

```

Two further tools. \bbbl@ifsamestring first expand its arguments and then compare their expansion (sanitized, so that the catcodes do not matter). \bbbl@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\bbbl@ifsamestring#1#2{%
141   \begingroup
142     \protected@edef\bbbl@tempb{#1}%
143     \edef\bbbl@tempb{\expandafter\strip@prefix\meaning\bbbl@tempb}%
144     \protected@edef\bbbl@tempc{#2}%
145     \edef\bbbl@tempc{\expandafter\strip@prefix\meaning\bbbl@tempc}%
146     \ifx\bbbl@tempb\bbbl@tempc
147       \aftergroup\@firstoftwo
148     \else
149       \aftergroup\@secondoftwo
150     \fi
151   \endgroup}
152 \chardef\bbbl@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\bbbl@bsphack{%
163 \ifhmode
164 \hskip\z@skip
165 \def\bbbl@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
166 \else
167 \let\bbbl@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\bbbl@cased{%
170 \ifx\oe\OE
171 \expandafter\in@\expandafter
172 {\expandafter\OE\expandafter}\expandafter{\oe}%
173 \ifin@
174 \bbbl@afterelse\expandafter\MakeUppercase
175 \else
176 \bbbl@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\bbbl@extras@wrap#1#2#3{% 1:in-test, 2:before, 3:after
182 \toks@\expandafter\expandafter\expandafter{%
183 \csname extras\language\endcsname}%
184 \bbbl@exp{\in@{#1}}{\the\toks@}}%
185 \ifin@\else
186 \@temptokena{#2}%
187 \edef\bbbl@tempc{\the\@temptokena\the\toks@}%
188 \toks@\expandafter{\bbbl@tempc#3}%
189 \expandafter\edef\csname extras\language\endcsname{\the\toks@}%
190 \fi}
191 <</Basic macros>>

```

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

```

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

```

3.1 Multiple languages

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

```

199 <<*Define core switching macros>> ≡

```



```

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

```

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

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

```

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

```

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

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

```

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

```

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

```

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

```

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

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

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

```

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

```

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

```

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

```

3.3 base

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

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

```

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

```

3.4 key=value options and other general option

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

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

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

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

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

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

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

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

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

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

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

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

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

3.5 Conditional loading of shorthands

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

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

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

```

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

```

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

```

396 \def\bbbl@ifshorthand#1{%
397 \bbbl@xin@\string#1}{\bbbl@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\bbbl@opt@shorthands{%
404 \expandafter\bbbl@sh@string\bbbl@opt@shorthands\@empty}%

```

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

```

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

```

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

```

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

```

For the option safe we use a different approach – \bbbl@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\bbbl@opt@safe\@undefined
417 \def\bbbl@opt@safe{BR}
418 % \let\bbbl@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 \bbbl@trace{Defining IfBabelLayout}
421 \ifx\bbbl@opt@layout\@nnil
422 \newcommand\IfBabelLayout[3]{#3}%
423 \else
424 \bbbl@exp{\bbbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
425 \in@{,layout,}{, #1,}%
426 \ifin@
427 \def\bbbl@opt@layout{#2}%
428 \bbbl@replace\bbbl@opt@layout{ }{.}%
429 \fi}
430 \newcommand\IfBabelLayout[1]{%
431 \expandtwoargs\in@{.#1.}{.\bbbl@opt@layout.}%
432 \ifin@
433 \expandafter\@firstoftwo
434 \else

```

```

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

```

3.6 Interlude for Plain

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

```

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

```

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

```

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

```

4 Multiple languages

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

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

```

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

```

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

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

```

468 \def\bb@fixname#1{%
469   \begingroup
470   \def\bb@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{l@#1}{\@no lanerr{#1}\@gobble}\@firstofone}

```

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

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

```

486 \def\bbl@bcpcase#1#2#3#4\@#5{%
487   \ifx\@empty#3%
488     \uppercase{\def#5{#1#2}}%
489   \else
490     \uppercase{\def#5{#1}}%
491     \lowercase{\edef#5{#5#2#3#4}}%
492   \fi}
493 \def\bbl@bcpllookup#1-#2-#3-#4\@@{%
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\@@\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\@@\bbl@tempb
508     \bbl@bcpcase#3\@empty\@empty\@@\bbl@tempc
509     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
510     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
511     {}%
512     \ifx\bbl@bcp\relax
513       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
514       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
515       {}%
516     \fi
517     \ifx\bbl@bcp\relax
518       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
519       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
520       {}%
521     \fi
522     \ifx\bbl@bcp\relax
523       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
524     \fi
525   \fi\fi}
526 \let\bbl@initload\relax
527 (-core)

```

```

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

```

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

```

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

```

4.1 Selecting the language

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

```

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

```

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

```

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

```

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

```

573 \let\xstring\string

```


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

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

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

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

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

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

`\bbl@pop@language`

```
575 \def\bbl@push@language{%
576   \ifx\language\@undefined\else
577     \ifx\currentgrouplevel\@undefined
578       \xdef\bbl@language@stack{\language+\bbl@language@stack}%
579     \else
580       \ifnum\currentgrouplevel=\z@
581         \xdef\bbl@language@stack{\language+}%
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@@\languagename}\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 = '\languagename'
610 }%
611 \fi}%
612 {}%
613 \chardef\localeid\bbl@c{l{id@}}

```

The unprotected part of `\selectlanguage`.

```

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

```

`\bbl@set@language` The macro `\bbl@set@language` takes care of switching the language environment *and* of writing entries on the auxiliary files. For historical reasons, language names can be either `language` or `\languagename`. To catch either form a trick is used, but unfortunately as a side effect the catcodes of letters in `\languagename` 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\languagename{%
623 \ifnum\escapechar=\expandafter`\string#1\@empty
624 \else\string#1\@empty\fi}%
625 \ifcat\relax\noexpand#1%
626 \expandafter\ifx\csname date\languagename\endcsname\relax
627 \edef\languagename{#1}%
628 \let\localename\languagename
629 \else
630 \bbl@info{Using '\string\languagename' instead of 'language' is\\%
631 deprecated. If what you want is to use a\\%
632 macro containing the actual locale, make\\%
633 sure it does not not match any language.\\%
634 Reported}%
635 \ifx\scantokens\@undefined
636 \def\localename{??}%
637 \else
638 \scantokens\expandafter{\expandafter
639 \def\expandafter\localename\expandafter{\languagename}}%
640 \fi
641 \fi
642 \else
643 \def\localename{#1}% This one has the correct catcodes
644 \fi
645 \select@language{\languagename}%
646 % write to auxs
647 \expandafter\ifx\csname date\languagename\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@bcppallowed
662 \bbl@bcppallowedfalse
663 \def\select@language#1{% from set@, babel@aux
664     \ifx\bbl@selectorname\@empty
665         \def\bbl@selectorname{select}%
666     % set hymap
667     \fi
668     \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
669     % set name
670     \edef\languagename{#1}%
671     \bbl@fixname\languagename
672     % TODO. name@map must be here?
673     \bbl@provide@locale
674     \bbl@iflanguage\languagename{%
675         \let\bbl@select@type\z@
676         \expandafter\bbl@switch\expandafter{\languagename}}
677 \def\babel@aux#1#2{%
678     \select@language{#1}%
679     \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
680         \writefile{##1}{\babel@toc{#1}{#2}\relax}}% TODO - plain?
681 \def\babel@toc#1#2{%
682     \select@language{#1}}

```

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

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

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

The switching of the values of `\lefthyphenmin` and `\righthyphenmin` is somewhat different. First we save their current values, then we check if `\langle lang \rangle hyphenmins` is defined. If it is not, we set default values (2 and 3), otherwise the values in `\langle lang \rangle 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\languagename @bbl@hyphenmap\endcsname
729 \fi
730 \chardef\bbl@opt@hyphenmap\z@
731 \else
732 \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
733 \csname\languagename @bbl@hyphenmap\endcsname
734 \fi
735 \fi
736 \let\bbl@hymapsel\@cclv
737 % hyphenation - select rules
738 \ifnum\csname l@\languagename\endcsname=\l@unhyphenated
739 \edef\bbl@tempa{u}%
740 \else
741 \edef\bbl@tempa{\bbl@ccl{\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=\@ccclv\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=\@ccclv\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\bbbl@beforeforeign{}
787 \edef\foreignlanguage{%
788   \noexpand\protect
789   \expandafter\noexpand\csname foreignlanguage \endcsname}
790 \expandafter\def\csname foreignlanguage \endcsname{%
791   \@ifstar\bbbl@foreign@s\bbbl@foreign@x}
792 \providecommand\bbbl@foreign@x[3][]{%
793   \begingroup
794     \def\bbbl@selectorname{foreign}%
795     \def\bbbl@select@opts{#1}%
796     \let\BabelText\@firstofone
797     \bbbl@beforeforeign
798     \foreign@language{#2}%
799     \bbbl@usehooks{foreign}{}%
800     \BabelText{#3}% Now in horizontal mode!
801   \endgroup}
802 \def\bbbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
803   \begingroup
804     {\par}%
805     \def\bbbl@selectorname{foreign*}%
806     \let\bbbl@select@opts\@empty
807     \let\BabelText\@firstofone
808     \foreign@language{#1}%
809     \bbbl@usehooks{foreign*}{}%
810     \bbbl@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 `bbbl@switch`.

```

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

```

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

```

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

```

`\bbbl@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). `\bbbl@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\bb@hyphlist\@empty
835 \let\bb@hyphenation@\relax
836 \let\bb@pttnlist\@empty
837 \let\bb@patterns@relax
838 \let\bb@hymapsel=\@cclv
839 \def\bb@patterns#1{%
840   \language=\expandafter\ifx\csname l@#1:f@encoding\endcsname\relax
841     \csname l@#1\endcsname
842     \edef\bb@tempa{#1}%
843   \else
844     \csname l@#1:f@encoding\endcsname
845     \edef\bb@tempa{#1:f@encoding}%
846   \fi
847   \@expandtwoargs\bb@usehooks{patterns}{#1}{\bb@tempa}}%
848 % > luatex
849 \@ifundefined{bb@hyphenation@}{% Can be \relax!
850   \begingroup
851     \bb@xin@{, \number\language,}{, \bb@hyphlist}%
852     \ifin@else
853       \@expandtwoargs\bb@usehooks{hyphenation}{#1}{\bb@tempa}}%
854     \hyphenation{%
855       \bb@hyphenation@
856       \@ifundefined{bb@hyphenation@#1}%
857         \@empty
858         {\space\csname bb@hyphenation@#1\endcsname}}%
859     \xdef\bb@hyphlist{\bb@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\bb@tempf{#1}%
864   \bb@fixname\bb@tempf
865   \bb@iflanguage\bb@tempf{%
866     \expandafter\bb@patterns\expandafter{\bb@tempf}%
867     \ifx\languageshorthands\@undefined\else
868       \languageshorthands{none}%
869     \fi
870     \expandafter\ifx\csname\bb@tempf hyphenmins\endcsname\relax
871       \set@hyphenmins\tw@\thr@@\relax
872     \else
873       \expandafter\expandafter\expandafter\set@hyphenmins
874       \csname\bb@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 `\langhyphenmins` is already defined this command has no effect.

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

4.2 Errors

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

`\@noopterr` When the package was loaded without options not everything will work as expected. An error message is issued in that case. When the format knows about `\PackageError` it must be $\text{\LaTeX} 2_{\epsilon}$, so we can safely use its error handling interface. Otherwise we’ll have to ‘keep it simple’. Infos are not written to the console, but on the other hand many people think warnings are errors, so a further message type is defined: an important info which is sent to the console.

```

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

```



```

917 \edef\bbbl@tempa{#1}%
918 \bbbl@sreplace\bbbl@tempa{name}{}%
919 \bbbl@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}{\bbbl@tempa}{..}\\%
924   Feel free to contribute on github.com/latex3/babel.\\%
925   Reported}}
926 \def\bbbl@tentative{\protect\bbbl@tentative@i}
927 \def\bbbl@tentative@i#1{%
928   \bbbl@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   \bbbl@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   \bbbl@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 \bbbl@nulllanguage\space instead}}
946 \let\bbbl@usehooks\@gobbletwo
947 \ifx\bbbl@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\bbbl@luapatterns\@undefined
951     \input luababel.def
952   \fi
953 \fi
954 \bbbl@trace{Compatibility with language.def}
955 \ifx\bbbl@languages\@undefined
956   \ifx\directlua\@undefined
957     \openin1 = language.def % TODO. Remove hardcoded number
958     \ifeof1
959       \closein1
960       \message{I couldn't find the file language.def}
961     \else
962       \closein1
963       \begingroup
964         \def\addlanguage#1#2#3#4#5{%
965           \expandafter\ifx\csname lang@#1\endcsname\relax\else
966             \global\expandafter\let\csname l@#1\expandafter\endcsname
967             \csname lang@#1\endcsname
968           \fi}%
969         \def\uselanguage#1{%
970           \input language.def
971         \endgroup
972       \fi
973     \fi
974   \chardef\l@english\z@
975 \fi

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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. `\bb@usehooks` is the commands used by `babel` to execute hooks defined for an event.

```

1009 \bb@trace{Hooks}
1010 \newcommand\AddBabelHook[3][ ]{%
1011   \bb@ifunset{\bb@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\languagename}
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@}%
1026     \ifx\languagename\@undefined\else % Test required for Plain (?)
1027       \ifx\UseHook\@undefined\else\UseHook{babel/#1/#2}\fi
1028       \def\bbl@elth##1{%
1029         \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#1#3}}}%
1030         \bbl@cs{ev@#2@#1}%
1031       \fi}

```

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

```

1032 \def\bbl@evargs{,% <- don't delete this comma
1033   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1034   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1035   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1036   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1037   beforestart=0,languagename=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@screaset\@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`\  

1120   \chardef\eqcatcode=\catcode`\  

1121   \catcode`\  

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`\  

1136   \catcode`\  

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@screaset\relax}%
1143 \def\ldf@finish#1{%
1144   \loadlocalcfg{#1}%
1145   \bbl@afterldf{#1}%
1146   \expandafter\main@language\expandafter{#1}%
1147   \catcode`\  

1148   \catcode`\  


```

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

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

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

4.5 Shorthands

`\bbl@add@special` The macro `\bbl@add@special` is used to add a new character (or single character control sequence) to the macro `\dospecials` (and `\@sanitize` if \TeX is used). It is used only at one place, namely

when `\initiate@active@char` is called (which is ignored if the char has been made active before). Because `\@sanitize` can be undefined, we put the definition inside a conditional. Items are added to the lists without checking its existence or the original catcode. It does not hurt, but should be fixed. It's already done with `\nfss@catcodes`, added in 3.10.

```

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

```

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

```

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

```

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

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

```

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

```

When there is also no current-level shorthand with an argument we will check whether there is a next-level defined shorthand for this active character.

```

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

```

\initiate@active@char calls \@initiate@active@char with 3 arguments. All of them are the same character with different catcodes: active, other (\string'ed) and the original one. This trick simplifies the code a lot.

```

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

```

The very first thing to do is saving the original catcode and the original definition, even if not active, which is possible (undefined characters require a special treatment to avoid making them \relax and preserving some degree of protection).

```

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

```

If the character is already active we provide the default expansion under this shorthand mechanism. Otherwise we write a message in the transcript file, and define \normal@char⟨char⟩ to expand to the character in its default state. If the character is mathematically active when babel is loaded (for example ') the normal expansion is somewhat different to avoid an infinite loop (but it does not prevent the loop if the mathcode is set to "8000 *a posteriori*).

```

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

```

To prevent problems with the loading of other packages after babel we reset the catcode of the character to the original one at the end of the package and of each language file (except with KeepShorthandsActive). It is re-activate again at \begin{document}. We also need to make sure that the shorthands are active during the processing of the .aux file. Otherwise some citations may give unexpected results in the printout when a shorthand was used in the optional argument of \bibitem for example. Then we make it active (not strictly necessary, but done for backward compatibility).

```

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

```

Now we have set \normal@char⟨char⟩, we must define \active@char⟨char⟩, to be executed when the character is activated. We define the first level expansion of \active@char⟨char⟩ to check the

status of the `@safe@actives` flag. If it is set to true we expand to the ‘normal’ version of this character, otherwise we call `\user@active⟨char⟩` to start the search of a definition in the user, language and system levels (or eventually `normal@char⟨char⟩`).

```

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

```

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

$$\backslash active@prefix \langle char \rangle \backslash normal@char \langle char \rangle$$

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

```

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

```

The next level of the code checks whether a user has defined a shorthand for himself with this character. First we check for a single character shorthand. If that doesn’t exist we check for a shorthand with an argument.

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

Initiating a shorthand makes active the char. That is not strictly necessary but it is still done for backward compatibility. So we need to restore the original catcode at the end of package *and* the end of the ldf.

```

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

```

`\bbm@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 `\bbm@firstcs` or `\bbm@scndcs`. Hence two more arguments need to follow it.

```

1327 \def\bbm@sh@select#1#2{%
1328   \expandafter\ifx\csname#1@sh@#2@sel@endcsname\relax
1329   \bbm@afterelse\bbm@scndcs
1330   \else
1331   \bbm@afterfi\csname#1@sh@#2@sel@endcsname
1332   \fi}

```

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

```

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

```

```

1359     \fi}}
1360 \endgroup

```

`\if@safe@actives` In some circumstances it is necessary to be able to reset the shorthand to its ‘normal’ value (usually the character with catcode ‘other’) on the fly. For this purpose the switch `@safe@actives` is available. The setting of this switch should be checked in the first level expansion of `\active@char⟨char⟩`. When this expansion mode is active (with `\@safe@activestruer`), 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).`

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

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

```

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

```

```

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

```

`\textormath` Some of the shorthands that will be declared by the language definition files have to be usable in both text and mathmode. To achieve this the helper macro `\textormath` is provided.

```

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

```

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

```

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

```

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

```

1418 \def\useshorthands{%
1419 \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}
1420 \def\bbl@usesh@s#1{%
1421 \bbl@usesh@x
1422 {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1423 {#1}}
1424 \def\bbl@usesh@x#1#2{%
1425 \bbl@ifshorthand{#2}%
1426 {\def\user@group{user}%
1427 \initiate@active@char{#2}%
1428 #1%
1429 \bbl@activate{#2}}%
1430 {\bbl@error
1431 {I can't declare a shorthand turned off (\string#2)}
1432 {Sorry, but you can't use shorthands which have been\\%
1433 turned off in the package options}}}

```

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

```

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

```

```

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

```

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

```

1453 \def\languageshorthands#1{\def\language@group{#1}}

```

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

```

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

```

`\notshorthand`

```

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

```

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

```

1477 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1478 \DeclareRobustCommand*\shorthandoff{%
1479   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1480 \def\bbl@shorthandoff#1#2{\bbl@switch@sh#1#2\@nnil}

```

`\bbl@switch@sh` The macro `\bbl@switch@sh` takes the list of characters apart one by one and subsequently switches the category code of the shorthand character according to the first argument of `\bbl@switch@sh`. But before any of this switching takes place we make sure that the character we are dealing with is known as a shorthand character. If it is, a macro such as `\active@char` should exist.

Switching off and on is easy – we just set the category code to ‘other’ (12) and \active. With the starred version, the original catcode and the original definition, saved in @initiate@active@char, are restored.

```

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

```

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

```

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

```

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

or off.

```
1537 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{bbl@active@\string#1}{#3}{#2}}
```

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

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

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

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

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

```
1560 \expandafter\def\csname OT1dqpos\endcsname{127}
1561 \expandafter\def\csname T1dqpos\endcsname{4}
```

When the macro `\f@encoding` is undefined (as it is in plain \TeX) we define it here to expand to OT1

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

4.6 Language attributes

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

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

```
1565 \bbl@trace{Language attributes}
1566 \newcommand\languageattribute[2]{%
1567   \def\bbl@tempc{#1}%
1568   \bbl@fixname\bbl@tempc
1569   \bbl@iflanguage\bbl@tempc{%
1570     \bbl@vforeach{#2}{%
```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

4.7 Support for saving macro definitions

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

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

```

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

```

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

```

1631 \newcount\babel@savecnt
1632 \babel@beginsave

```

`\babel@save` The macro `\babel@save{csname}` saves the current meaning of the control sequence `<csname>` to `\originalTeX`². To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to `\originalTeX` and the counter is incremented. The macro `\babel@savevariable{variable}` saves the value of the variable. `<variable>` can be anything allowed after the `\the` primitive. To avoid messing saved definitions up, they are saved only the very first time.

```

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

```

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

```

1645 \fi}
1646 \def\babel@savevariable#1{%
1647 \toks@\expandafter{\originalTeX #1=}%
1648 \bbl@exp{\def\originalTeX{\the\toks@\the#1\relax}}}

```

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

```

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

```

4.8 Short tags

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

```

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

```

```

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

```

4.9 Hyphens

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

```

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

```

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

```

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

```

`\babelhyphen` Macros to insert common hyphens. Note the space before `@` in `\babelhyphen`. Instead of protecting it with `\DeclareRobustCommand`, which could insert a `\relax`, we use the same procedure as shorthands, with `\active@prefix`.

```

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

```

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

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

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

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

The following macro inserts the hyphen char.

```
1743 \def\bb@hyphenchar{%
1744   \ifnum\hyphenchar\font=\m@ne
1745     \babe\nul\hyphen
1746   \else
1747     \char\hyphenchar\font
1748   \fi}
```

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

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

`\bb@disc` For some languages the macro `\bb@disc` is used to ease the insertion of discretionaries for letters that behave ‘abnormally’ at a breakpoint.

```
1763 \def\bb@disc#1#2{\nobreak\discretionary{#2-}{#1}\bb@allowhyphens}
```

4.10 Multiencoding strings

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

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

```
1764 \bb@trace{Multiencoding strings}
1765 \def\bb@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 `\bb@uclc`. The parser is restarted inside `\langle lang\rangle\bb@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\bb@tolower\@empty\bb@toupper\@empty
```

and starts over (and similarly when lowercasing).

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

```

1768 {\def\bb@patchuclc{% TODO. Delete. Doesn't work any more.
1769 \global\let\bb@patchuclc\relax
1770 \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bb@uclc}}%
1771 \gdef\bb@uclc##1{%
1772 \let\bb@encoded\bb@encoded@uclc
1773 \bb@ifunset{\languagename @bb@uclc}% and resumes it
1774 {##1}%
1775 {\let\bb@tempa##1\relax % Used by LANG@bb@uclc
1776 \csname\languagename @bb@uclc\endcsname}%
1777 {\bb@tolower\@empty}{\bb@toupper\@empty}}%
1778 \gdef\bb@tolower{\csname\languagename @bb@lc\endcsname}%
1779 \gdef\bb@toupper{\csname\languagename @bb@uc\endcsname}}
1780 <<(*More package options)>> ≡
1781 \DeclareOption{nocase}{}
1782 <</More package options>>

```

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

```

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

```

Main command This is the main command. With the first use it is redefined to omit the basic setup in subsequent blocks. We make sure strings contain actual letters in the range 128-255, not active characters.

```

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

```

```

1824 \def\bbL@startcmds@i#1#2{%
1825   \edef\bbL@L{\zap@space#1 \@empty}%
1826   \edef\bbL@G{\zap@space#2 \@empty}%
1827   \bbL@startcmds@ii}
1828 \let\bbL@startcommands\StartBabelCommands

```

Parse the encoding info to get the label, input, and font parts.

Select the behavior of \SetString. There are two main cases, depending of if there is an optional argument: without it and strings=encoded, strings are defined always; otherwise, they are set only if they are still undefined (ie, fallback values). With labelled blocks and strings=encoded, define the strings, but with another value, define strings only if the current label or font encoding is the value of strings; otherwise (ie, no strings or a block whose label is not in strings=) do nothing. We presume the current block is not loaded, and therefore set (above) a couple of default values to gobble the arguments. Then, these macros are redefined if necessary according to several parameters.

```

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

```

```

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

```

There are two versions of `\bbl@scswitch`. The first version is used when `\ldfs` are read, and it makes sure `\langle group \rangle \langle language \rangle` is reset, but only once (`\bbl@screset` is used to keep track of this). The second version is used in the preamble and packages loaded after `babel` and does nothing. The macro `\bbl@forlang` loops `\bbl@L` but its body is executed only if the value is in `\BabelLanguages` (inside `babel`) or `\date \langle language \rangle` is defined (after `babel` has been loaded). There are also two version of `\bbl@forlang`. The first one skips the current iteration if the language is not in `\BabelLanguages` (used in `\ldfs`), and the second one skips undefined languages (after `babel` has been loaded).

```

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

```

Now we define commands to be used inside `\StartBabelCommands`.

Strings The following macro is the actual definition of `\SetString` when it is “active”. First save the “switcher”. Create it if undefined. Strings are defined only if undefined (ie, like `\providescommand`). With the event `stringprocess` you can preprocess the string by manipulating the value of `\BabelString`. If there are several hooks assigned to this event, preprocessing is done in the same order as defined. Finally, the string is set.

```

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

```

```

1923 \expandafter\bbL@stringdef
1924 \csname\bbL@LC\expandafter\endcsname\expandafter{\BabelString}}

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

Macros to deal with case mapping for hyphenation. To decide if the document is monolingual or multilingual, we make a rough guess – just see if there is a comma in the languages list, built in the first pass of the package options.

```

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

```



```

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

```

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

```

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

```

The following package options control the behavior of hyphenation mapping.

```

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

```

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

```

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

```

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

```

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

```

```

2021     {\bbl@tempd}%
2022 \ifin@ % Renew caption
2023     \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2024     \ifin@
2025     \bbl@exp{%
2026         \\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2027         {\bbl@scset\<#2name>\<#1#2name>}%
2028         {}}%
2029 \else % Old way converts to new way
2030     \bbl@ifunset{#1#2name}%
2031     {\bbl@exp{%
2032         \\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2033         \\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2034         {\def\<#2name>{\<#1#2name>}}%
2035         {}}}%
2036     {}}%
2037 \fi
2038 \else
2039     \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2040     \ifin@ % New way
2041     \bbl@exp{%
2042         \\bbl@add\<captions#1>{\bbl@scset\<#2name>\<#1#2name>}%
2043         \\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2044         {\bbl@scset\<#2name>\<#1#2name>}%
2045         {}}%
2046 \else % Old way, but defined in the new way
2047     \bbl@exp{%
2048         \\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2049         \\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2050         {\def\<#2name>{\<#1#2name>}}%
2051         {}}%
2052 \fi%
2053 \fi
2054 \@namedef{#1#2name}{#3}%
2055 \toks@\expandafter{\bbl@captionslist}%
2056 \bbl@exp{\in@{\<#2name>}{\the\toks@}}%
2057 \ifin@ \else
2058     \bbl@exp{\bbl@add\bbl@captionslist{\<#2name>}}%
2059     \bbl@toggle\bbl@captionslist
2060 \fi
2061 \fi}
2062 % \def\bbl@setcaption@s#1#2#3{} % TODO. Not yet implemented (w/o 'name')

```

4.11 Macros common to a number of languages

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

```

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

```

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

```

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

```

4.12 Making glyphs available

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

```
2071 \ProvideTextCommand{\quotedblbase}{OT1}{%
2072   \save@sf@q{\set@low@box{\textquotedblright\}}%
2073   \box\z@\kern-.04em\bbl@allowhyphens}}
```

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

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

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

```
2076 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2077   \save@sf@q{\set@low@box{\textquoteright\}}%
2078   \box\z@\kern-.04em\bbl@allowhyphens}}
```

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

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

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

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

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

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

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

`\guilsinglright`

```

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

```

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

```

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

```

4.12.2 Letters

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

```

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

```

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

```

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

```

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

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

```

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

```

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

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

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

```
2168 \DeclareTextCommand{\SS}{OT1}{SS}
2169 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

4.12.3 Shorthands for quotation marks

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

`\glq` The ‘german’ single quotes.

```
\grq
2170 \ProvideTextCommandDefault{\glq}{%
2171   \textormath{\quotingsinglbase}{\mbox{\quotingsinglbase}}}
```

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

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

`\glqq` The ‘german’ double quotes.

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

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

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

`\flq` The ‘french’ single guillemets.

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

`\flqq` The ‘french’ double guillemets.

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

4.12.4 Umlauts and tremas

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

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

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

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

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

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

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

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

For all vowels we declare `\` to be a composite command which uses `\bbbl@umlauta` or `\bbbl@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 `\bbbl@umlauta` and/or `\bbbl@umlaute` for a language in the corresponding `ldf` (using the `babel` switching mechanism, of course).

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

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

```

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

```

4.13 Layout

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

```

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

```

4.14 Load engine specific macros

Some macros are not defined in all engines, so, after loading the files define them if necessary to raise an error:

```

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

```

```

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

```

4.15 Creating and modifying languages

Continue with \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.

```

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

```



```

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

```

```

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

```

At this point all parameters are defined if 'import'. Now we execute some code depending on them. But what about if nothing was imported? We just set the basic parameters, but still loading the whole ini file.

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

The hyphenrules option is handled with an auxiliary macro. This macro is called in three cases:

when a language is first declared with `\babelprovide`, with `hyphenrules` and with `import`.

```

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

```

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

```

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

```

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

```



```

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

```

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

```

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

```

```

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

```

A variant to be used when the ini file has been already loaded, because it's not the first \babelprovide for this language.

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

2965 \def\bb@exportkey#1#2#3{%
2966   \bb@ifunset{bb@kv@#2}%
2967   {\bb@csarg\gdef{#1@\languagename}{#3}}%
2968   {\expandafter\ifx\csname bb@kv@#2\endcsname\empty
2969     \bb@csarg\gdef{#1@\languagename}{#3}}%
2970   \else
2971     \bb@exp{\global\let<\bb@#1@\languagename><\bb@kv@#2>}%
2972   \fi}}

```

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

```

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

```

```

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

```

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

```

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

```

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

```

3039 \let\bbl@inikv@identification\bbl@inikv
3040 \let\bbl@inikv@date\bbl@inikv
3041 \let\bbl@inikv@typography\bbl@inikv
3042 \let\bbl@inikv@characters\bbl@inikv
3043 \let\bbl@inikv@numbers\bbl@inikv

```

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

```

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

```

```

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

```

Now captions and captions.licr, depending on the engine. And below also for dates. They rely on a few auxiliary macros. It is expected the ini file provides the complete set in Unicode and LICR, in that order:

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

To show correctly some captions in a few languages, we need to patch some internal macros, because the order is hardcoded. For example, in Japanese the chapter number is surrounded by two string, while in Hungarian is placed after. These replacement works in many classes, but not all. Actually, the following lines are somewhat tentative.

```

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

```

Date. Arguments (year, month, day) are *not* protected, on purpose. In `\today`, arguments are always gregorian, and therefore always converted with other calendars. TODO. Document

```

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

```

```

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

```

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

```

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

```



```

3279 \newcommand\BabelDateDot{\. \@} % TODO. \let instead of repeating
3280 \newcommand\BabelDated[1]{\number#1}
3281 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3282 \newcommand\BabelDateM[1]{\number#1}
3283 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3284 \newcommand\BabelDateMMM[1]{%
3285   \csname month\romannumeral#1\babel@calendar name\endcsname}%
3286 \newcommand\BabelDatey[1]{\number#1}%
3287 \newcommand\BabelDateyy[1]{%
3288   \ifnum#1<10 0\number#1 %
3289   \else\ifnum#1<100 \number#1 %
3290   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3291   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3292   \else
3293     \bbl@error
3294     {Currently two-digit years are restricted to the\
3295      range 0-9999.}%
3296     {There is little you can do. Sorry.}%
3297   \fi\fi\fi\fi}
3298 \newcommand\BabelDateyyyy[1]{\number#1} % TODO - add leading 0
3299 \newcommand\BabelDateU[1]{\number#1}%
3300 \def\bbl@replace@finish@iii#1{%
3301   \bbl@exp{\def\#1###1###2###3{\the\toks@}}
3302 \def\bbl@TG@date{%
3303   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3304   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3305   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{###3}}%
3306   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{###3}}%
3307   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{###2}}%
3308   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{###2}}%
3309   \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMM{###2}}%
3310   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{###1}}%
3311   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{###1}}%
3312   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{###1}}%
3313   \bbl@replace\bbl@toreplace{[U]}{\BabelDateU{###1}}%
3314   \bbl@replace\bbl@toreplace{[y|]}{\bbl@datecctr[###1|]}%
3315   \bbl@replace\bbl@toreplace{[U|]}{\bbl@datecctr[###1|]}%
3316   \bbl@replace\bbl@toreplace{[m|]}{\bbl@datecctr[###2|]}%
3317   \bbl@replace\bbl@toreplace{[d|]}{\bbl@datecctr[###3|]}%
3318   \bbl@replace@finish@iii\bbl@toreplace}
3319 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3320 \def\bbl@xdatecctr[#1|#2]{\localenumeral{#2}{#1}}

```

Transforms.

```

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

```

```

3340 \ifin@
3341 \bbl@foreach\bbl@KVP@transforms{&%
3342 \bbl@xin@{: \babeltempa,}{,##1,}&%
3343 \ifin@ &% font:font:transform syntax
3344 \directlua{
3345     local t = {}
3346     for m in string.gmatch('#1'..' ':'', '(.-):') do
3347         table.insert(t, m)
3348     end
3349     table.remove(t)
3350     token.set_macro('babeltempc', ', fonts=' .. table.concat(t, ' '))
3351 }&%
3352 \fi}&%
3353 \in@{.0$}{#2$}&%
3354 \ifin@
3355 \directlua{&% (\attribute) syntax
3356     local str = string.match([[ \bbl@KVP@transforms]],
3357         '%(([^%(-)]-)%[^\%)]-\babeltempa')
3358     if str == nil then
3359         token.set_macro('babeltempb', '')
3360     else
3361         token.set_macro('babeltempb', ', attribute=' .. str)
3362     end
3363 }&%
3364 \toks@{#3}&%
3365 \bbl@exp{&%
3366     \\g@addto@macro\\bbl@release@transforms{&%
3367     \relax &% Closes previous \bbl@transforms@aux
3368     \\bbl@transforms@aux
3369     \\#1{label=\babeltempa\babeltempb\babeltempc}&%
3370     {\languagename}{\the\toks@}}&%
3371 \else
3372 \g@addto@macro\bbl@release@transforms{, {#3}}&%
3373 \fi
3374 \fi}
3375 \endgroup

```

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

```

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

```

```

3400 \bbl@csarg\bbl@tglobal{lsys@#1}}
3401 \def\bbl@xenoHyph@d{%
3402 \bbl@ifset{bbl@prehc@\languagename}%
3403   {\ifnum\hyphenchar\font=\defaultHyphenchar
3404     \iffontchar\font\bbl@c{l{prehc}}\relax
3405     \hyphenchar\font\bbl@c{l{prehc}}\relax
3406     \else\iffontchar\font"200B
3407       \hyphenchar\font"200B
3408     \else
3409       \bbl@warning
3410       {Neither 0 nor ZERO WIDTH SPACE are available\\%
3411         in the current font, and therefore the hyphen\\%
3412         will be printed. Try changing the fontspec's\\%
3413         'HyphenChar' to another value, but be aware\\%
3414         this setting is not safe (see the manual).\\%
3415         Reported}%
3416       \hyphenchar\font\defaultHyphenchar
3417     \fi\fi
3418   \fi}%
3419   {\hyphenchar\font\defaultHyphenchar}}
3420 % \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).

```

3421 \def\bbl@load@info#1{%
3422 \def\BabelBeforeIni##1##2{%
3423   \beginGroup
3424     \bbl@read@ini{##1}0%
3425     \endinput          % babel- .tex may contain only preamble's
3426     \endGroup}%       boxed, to avoid extra spaces:
3427   {\bbl@input@texini{#1}}

```

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

```

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

```

```

3455     \\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi
3456     \\expandafter\<bbl@digits@\language\>%
3457     \\\fi}}}%
3458     \bbl@tempa}

```

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

```

3459 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={
3460     \ifx\\#1%           % \\ before, in case #1 is multiletter
3461     \bbl@exp{%
3462     \def\\bbl@tempa###1{%
3463     \<ifcase>###1\space\the\toks@\<else>\\@ctrerr\<fi>}}%
3464     \else
3465     \toks@\expandafter{\the\toks@\or #1}%
3466     \expandafter\bbl@buildifcase
3467     \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 a special case, for a fixed form (see babel-he.ini, for example).

```

3468 \newcommand\localenumerat[2]{\bbl@cs{cntr@#1@\language}\#2}}
3469 \def\bbl@localecntr#1#2{\localenumerat{#2}{#1}}
3470 \newcommand\localecounter[2]{%
3471     \expandafter\bbl@localecntr
3472     \expandafter{\number\csname c@#2\endcsname}{#1}}
3473 \def\bbl@alphanumeric#1#2{%
3474     \expandafter\bbl@alphanumeric@i\number#2 76543210\@{#1}}
3475 \def\bbl@alphanumeric@i#1#2#3#4#5#6#7#8\@{#9}{%
3476     \ifcase\@car#8\@nil\or % Currenty <10000, but prepared for bigger
3477     \bbl@alphanumeric@ii{#9}00000#1\or
3478     \bbl@alphanumeric@ii{#9}00000#1#2\or
3479     \bbl@alphanumeric@ii{#9}00000#1#2#3\or
3480     \bbl@alphanumeric@ii{#9}00000#1#2#3#4\else
3481     \bbl@alphanum@invalid{>9999}%
3482     \fi}
3483 \def\bbl@alphanumeric@ii#1#2#3#4#5#6#7#8{%
3484     \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\language}%
3485     {\bbl@cs{cntr@#1.4@\language}\#5%
3486     \bbl@cs{cntr@#1.3@\language}\#6%
3487     \bbl@cs{cntr@#1.2@\language}\#7%
3488     \bbl@cs{cntr@#1.1@\language}\#8%
3489     \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3490     \bbl@ifunset{bbl@cntr@#1.S.321@\language}}}%
3491     {\bbl@cs{cntr@#1.S.321@\language}}}%
3492     \fi}%
3493     {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}}}}
3494 \def\bbl@alphanum@invalid#1{%
3495     \bbl@error{Alphabetic numeral too large (#1)}%
3496     {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.

```

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

```

```

3508             Perhaps it doesn't exist}%
3509             {See the manual for details.}}%
3510         {#1}%
3511     \fi}
3512 % \@namedef{bbl@info@name.locale}{lcname}
3513 \@namedef{bbl@info@tag.ini}{lini}
3514 \@namedef{bbl@info@name.english}{elname}
3515 \@namedef{bbl@info@name.opentype}{lname}
3516 \@namedef{bbl@info@tag.bcp47}{tbcpl}
3517 \@namedef{bbl@info@language.tag.bcp47}{lbcpl}
3518 \@namedef{bbl@info@tag.opentype}{lotf}
3519 \@namedef{bbl@info@script.name}{esname}
3520 \@namedef{bbl@info@script.name.opentype}{sname}
3521 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3522 \@namedef{bbl@info@script.tag.opentype}{sotf}
3523 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3524 \@namedef{bbl@info@variant.tag.bcp47}{vbcpl}
3525 \@namedef{bbl@info@extension.t.tag.bcp47}{extt}
3526 \@namedef{bbl@info@extension.u.tag.bcp47}{extu}
3527 \@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.

```

3528 \providecommand\BCPdata{}
3529 \ifx\renewcommand\undefined\else % For plain. TODO. It's a quick fix
3530   \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty}
3531   \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3532     \nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3533     {\bbl@bcpdata@ii#6}\bbl@main@language}%
3534     {\bbl@bcpdata@ii#1#2#3#4#5#6}\languagename}}%
3535   \def\bbl@bcpdata@ii#1#2{%
3536     \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3537     {\bbl@error{Unknown field '#1' in \string\BCPdata.\\%
3538       Perhaps you misspelled it.}%
3539     {See the manual for details.}}%
3540     {\bbl@ifunset{bbl@csname bbl@info@#1.tag.bcp47\endcsname @#2}{}%
3541     {\bbl@cs{csname bbl@info@#1.tag.bcp47\endcsname @#2}}}}
3542 \fi
3543 % Still somewhat hackish. WIP.
3544 \@namedef{bbl@info@casing.tag.bcp47}{casing}
3545 \newcommand\BabelUppercaseMapping[3]{%
3546   \let\bbl@tempx\languagename
3547   \edef\languagename{#1}%
3548   \DeclareUppercaseMapping[\BCPdata{casing}]{#2}{#3}%
3549   \let\languagename\bbl@tempx}
3550 \newcommand\BabelLowercaseMapping[3]{%
3551   \let\bbl@tempx\languagename
3552   \edef\languagename{#1}%
3553   \DeclareLowercaseMapping[\BCPdata{casing}]{#2}{#3}%
3554   \let\languagename\bbl@tempx}

```

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

```

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

```

```

3566 \def\languagename{##1}%
3567 \bbl@ensureinfo{##1}}}}
3568 \@ifpackagewith{babel}{ensureinfo=off}{}%
3569 {\AtEndOfPackage{% Test for plain.
3570 \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`.

```

3571 \newcommand\getlocaleproperty{%
3572 \ifstar\bbl@getproperty@s\bbl@getproperty@x}
3573 \def\bbl@getproperty@s#1#2#3{%
3574 \let#1\relax
3575 \def\bbl@elt##1##2##3{%
3576 \bbl@ifsamestring{##1/##2}{##3}%
3577 {\providecommand#1{##3}%
3578 \def\bbl@elt###1###2###3{}}}%
3579 {}}%
3580 \bbl@cs{inidata@#2}}%
3581 \def\bbl@getproperty@x#1#2#3{%
3582 \bbl@getproperty@s{#1}{#2}{#3}%
3583 \ifx#1\relax
3584 \bbl@error
3585 {Unknown key for locale '#2':\%
3586 #3\%
3587 \string#1 will be set to \relax}%
3588 {Perhaps you misspelled it.}%
3589 \fi}
3590 \let\bbl@ini@loaded@empty
3591 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

5 Adjusting the Babel behavior

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

```

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

```

```

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

```

```

3683 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3684   \directlua{ Babel.ignore_pre_char = function(node)
3685     return (node.lang == \the\csname l@nohyphenation\endcsname)
3686   end }}
3687 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3688   \directlua{ Babel.ignore_pre_char = function(node)
3689     return false
3690   end }}
3691 \@namedef{bbl@ADJ@select.write@shift}{%
3692   \let\bbl@restorelastskip\relax
3693   \def\bbl@savelastskip{%
3694     \let\bbl@restorelastskip\relax
3695     \ifvmode
3696       \ifdim\lastskip=\z@
3697         \let\bbl@restorelastskip\nobreak
3698       \else
3699         \bbl@exp{%
3700           \def\\bbl@restorelastskip{%
3701             \skip@=\the\lastskip
3702             \\nobreak \vskip-\skip@ \vskip\skip@}}%
3703         \fi
3704       \fi}}
3705 \@namedef{bbl@ADJ@select.write@keep}{%
3706   \let\bbl@restorelastskip\relax
3707   \let\bbl@savelastskip\relax}
3708 \@namedef{bbl@ADJ@select.write@omit}{%
3709   \AddBabelHook{babel-select}{beforestart}{%
3710     \expandafter\babel@aux\expandafter{\bbl@main@language}}}%
3711 \let\bbl@restorelastskip\relax
3712 \def\bbl@savelastskip##1\bbl@restorelastskip{}
3713 \@namedef{bbl@ADJ@select.encoding@off}{%
3714   \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.

```

3715 << *More package options >> ≡
3716 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3717 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3718 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3719 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3720 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3721 << /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.

```

3722 \bbl@trace{Cross referencing macros}
3723 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3724   \def\@newl@bel#1#2#3{%
3725     {\@safe@activetrue
3726       \bbl@ifunset{#1#2}%
3727       \relax
3728       {\gdef\@multiplelabels{%

```



```

3729     \@latex@warning@no@line{There were multiply-defined labels}}%
3730     \@latex@warning@no@line{Label `#2' multiply defined}}%
3731     \global\@namedef{#1@#2}{#3}}

```

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

```

3732 \CheckCommand*\@testdef[3]{%
3733   \def\reserved@a{#3}%
3734   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3735   \else
3736     \@tempwattrue
3737   \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.

```

3738 \def\@testdef#1#2#3{% TODO. With @samestring?
3739   \@safe@activestruetrue
3740   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3741   \def\bbl@tempb{#3}%
3742   \@safe@activesfalse
3743   \ifx\bbl@tempa\relax
3744   \else
3745     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3746   \fi
3747   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3748   \ifx\bbl@tempa\bbl@tempb
3749   \else
3750     \@tempwattrue
3751   \fi}
3752 \fi

```

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

```

3753 \bbl@xin@{R}\bbl@opt@safe
3754 \ifin@
3755 \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3756 \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3757   {\expandafter\strip@prefix\meaning\ref}%
3758 \ifin@
3759   \bbl@redefine\@kernel@ref#1{%
3760     \@safe@activestruetrue\org@@kernel@ref{#1}\@safe@activesfalse}
3761   \bbl@redefine\@kernel@pageref#1{%
3762     \@safe@activestruetrue\org@@kernel@pageref{#1}\@safe@activesfalse}
3763   \bbl@redefine\@kernel@sref#1{%
3764     \@safe@activestruetrue\org@@kernel@sref{#1}\@safe@activesfalse}
3765   \bbl@redefine\@kernel@spageref#1{%
3766     \@safe@activestruetrue\org@@kernel@spageref{#1}\@safe@activesfalse}
3767   \else
3768     \bbl@redefineroobust\ref#1{%
3769       \@safe@activestruetrue\org@ref{#1}\@safe@activesfalse}
3770     \bbl@redefineroobust\pageref#1{%
3771       \@safe@activestruetrue\org@pageref{#1}\@safe@activesfalse}
3772   \fi
3773 \else
3774   \let\org@ref\ref
3775   \let\org@pageref\pageref
3776 \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.

```
3777 \bbl@xin@{B}\bbl@opt@safe
3778 \ifin@
3779 \bbl@redefine\@citex[#1]#2{%
3780 \@safe@activestruedef\@tempa{#2}\@safe@activesfalse
3781 \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.

```
3782 \AtBeginDocument{%
3783 \@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.)

```
3784 \def\@citex[#1][#2]#3{%
3785 \@safe@activestruedef\@tempa{#3}\@safe@activesfalse
3786 \org@citex[#1][#2]{\@tempa}}%
3787 }{}}
```

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

```
3788 \AtBeginDocument{%
3789 \@ifpackageloaded{cite}{%
3790 \def\@citex[#1]#2{%
3791 \@safe@activestruedef\org@citex[#1][#2]\@safe@activesfalse}%
3792 }{}}
```

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

```
3793 \bbl@redefine\nocite#1{%
3794 \@safe@activestruedef\org@nocite[#1]\@safe@activesfalse}
```

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

```
3795 \bbl@redefine\bibcite{%
3796 \bbl@cite@choice
3797 \bibcite}
```

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

```
3798 \def\bbl@bibcite#1#2{%
3799 \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.

```
3800 \def\bbl@cite@choice{%
3801 \global\let\bibcite\bbl@bibcite
3802 \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
3803 \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{%
3804 \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.

```
3805 \AtBeginDocument{\bbl@cite@choice}
```

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

```

3806 \bbl@redefine\@bibitem#1{%
3807   \@safe@activestrue\org@bibitem{#1}\@safe@activesfalse}
3808 \else
3809   \let\org@nocite\nocite
3810   \let\org@citex@citex
3811   \let\org@bibcite\bibcite
3812   \let\org@bibitem@bibitem
3813 \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.

```

3814 \bbl@trace{Marks}
3815 \IfBabelLayout{sectioning}
3816   {\ifx\bbl@opt@headfoot\@nnil
3817     \g@addto@macro\@resetactivechars{%
3818       \set@typeset@protect
3819       \expandafter\select@language@x\expandafter{\bbl@main@language}%
3820       \let\protect\noexpand
3821       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3822         \edef\thepage{%
3823           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3824       \fi}%
3825   \fi}
3826 {\ifbbl@single\else
3827   \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
3828   \markright#1{%
3829     \bbl@ifblank{#1}%
3830     {\org@markright{}}%
3831     {\toks@{#1}%
3832       \bbl@exp{%
3833         \\org@markright{\\protect\\foreignlanguage{\language}%
3834           {\protect\\bbl@restore@actives\the\toks@}}}%

```

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

```

3835   \ifx\@mkboth\markboth
3836     \def\bbl@tempc{\let\@mkboth\markboth}%
3837   \else
3838     \def\bbl@tempc{%
3839       \fi
3840       \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
3841       \markboth#1#2{%
3842         \protected@edef\bbl@tempb##1{%
3843           \protect\foreignlanguage
3844             {\language}{\protect\bbl@restore@actives##1}}%
3845         \bbl@ifblank{#1}%
3846         {\toks@{}}%
3847         {\toks@\expandafter{\bbl@tempb{#1}}}%
3848         \bbl@ifblank{#2}%
3849         {\@temptokena{}}%
3850         {\@temptokena\expandafter{\bbl@tempb{#2}}}%

```

```

3851     \bbl@exp{\@org@markboth{\the\toks@}{\the\@temptokena}}}%
3852     \bbl@tempc
3853     \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.

```

3854 \bbl@trace{Preventing clashes with other packages}
3855 \ifx\org@ref\@undefined\else
3856   \bbl@xin@{R}\bbl@opt@safe
3857   \ifin@
3858     \AtBeginDocument{%
3859       \@ifpackageloaded{ifthen}{%
3860         \bbl@redefine@long\ifthenelse#1#2#3{%
3861           \let\bbl@temp@pref\pageref
3862           \let\pageref\org@pageref
3863           \let\bbl@temp@ref\ref
3864           \let\ref\org@ref
3865           \@safe@activestrue
3866           \org@ifthenelse{#1}%
3867             {\let\pageref\bbl@temp@pref
3868              \let\ref\bbl@temp@ref
3869              \@safe@activesfalse
3870              #2}%
3871             {\let\pageref\bbl@temp@pref
3872              \let\ref\bbl@temp@ref
3873              \@safe@activesfalse
3874              #3}%
3875           }%
3876         }{}%
3877       }
3878 \fi

```

5.3.2 varioref

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

```

3879 \AtBeginDocument{%
3880   \@ifpackageloaded{varioref}{%
3881     \bbl@redefine\@vppageref#1[#2]#3{%
3882       \@safe@activestrue
3883       \org@@@vppageref{#1}[#2]#3}%
3884       \@safe@activesfalse}%
3885     \bbl@redefine\vrefpagemum#1#2{%
3886       \@safe@activestrue

```

```

3887 \org@vrefpagenum{#1}{#2}%
3888 \@safe@activesfalse}%

```

The package `varioref` defines `\Ref` to be a robust command which uppercases the first character of the reference text. In order to be able to do that it needs to access the expandable form of `\ref`. So we employ a little trick here. We redefine the (internal) command `\Refl` 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.

```

3889 \expandafter\def\csname Ref \endcsname#1{%
3890 \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3891 }{}%
3892 }
3893 \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.

```

3894 \AtEndOfPackage{%
3895 \AtBeginDocument{%
3896 \ifpackageloaded{hhline}%
3897 {\expandafter\ifx\csname normal@char\string:\endcsname\relax
3898 \else
3899 \makeatletter
3900 \def\@currname{hhline}\input{hhline.sty}\makeatother
3901 \fi}%
3902 }}}

```

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

```

3903 \def\substitutefontfamily#1#2#3{%
3904 \lowercase{\immediate\openout15=#1#2.fd\relax}%
3905 \immediate\write15{%
3906 \string\ProvidesFile{#1#2.fd}%
3907 [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3908 \space generated font description file]^^J
3909 \string\DeclareFontFamily{#1}{#2}{}^^J
3910 \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
3911 \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
3912 \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
3913 \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
3914 \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
3915 \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
3916 \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
3917 \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
3918 }%
3919 \closeout15
3920 }
3921 \@onlypreamble\substitutefontfamily

```

5.4 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of \TeX and \LaTeX always come out in the right encoding. There is a list of non-ASCII encodings. Requested encodings are currently stored in `\@fontenc@load@list`. If a non-ASCII has been loaded, we define versions of `\TeX` and `\LaTeX` for them using `\ensureascii`. The default ASCII encoding is set, too (in reverse order): the “main” encoding (when the document begins), the last loaded, or OT1.

\ensureascii

```
3922 \bbl@trace{Encoding and fonts}
3923 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3924 \newcommand\BabelNonText{TS1,T3,TS3}
3925 \let\org@TeX\TeX
3926 \let\org@LaTeX\LaTeX
3927 \let\ensureascii\@firstofone
3928 \AtBeginDocument{%
3929   \def\@elt#1{,#1,}%
3930   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3931   \let\@elt\relax
3932   \let\bbl@tempb\@empty
3933   \def\bbl@tempc{OT1}%
3934   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3935     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3936   \bbl@foreach\bbl@tempa{%
3937     \bbl@xin@{#1}{\BabelNonASCII}%
3938     \ifin@
3939       \def\bbl@tempb{#1}% Store last non-ascii
3940     \else\bbl@xin@{#1}{\BabelNonText}% Pass
3941     \ifin@\else
3942       \def\bbl@tempc{#1}% Store last ascii
3943     \fi
3944     \fi}%
3945   \ifx\bbl@tempb\@empty\else
3946     \bbl@xin@{\cf@encoding,}{\BabelNonASCII,\BabelNonText,}%
3947     \ifin@\else
3948       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3949     \fi
3950     \edef\ensureascii#1{%
3951       {\noexpand\fontencoding{\bbl@tempc}\noexpand\selectfont#1}}%
3952     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3953     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3954   \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.

```
3955 \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.

```
3956 \AtBeginDocument{%
3957   \ifpackageloaded{fontspec}%
3958     {\xdef\latinencoding{%
3959       \ifx\UTFencname\undefined
3960         EU\ifcase\bbl@engine\or2\or1\fi
3961       \else
3962         \UTFencname
3963       \fi}}%
3964     {\gdef\latinencoding{OT1}%
3965       \ifx\cf@encoding\bbl@t@one
3966         \xdef\latinencoding{\bbl@t@one}%
3967       \else
3968         \def\@elt#1{,#1,}%
3969         \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3970         \let\@elt\relax
3971         \bbl@xin@{,T1,}\bbl@tempa
```

```

3972     \ifin@
3973     \xdef\latinencoding{\bbl@t@one}%
3974     \fi
3975     \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.

```

3976 \DeclareRobustCommand{\latintext}{%
3977   \fontencoding{\latinencoding}\selectfont
3978   \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.

```

3979 \ifx\@undefined\DeclareTextFontCommand
3980   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3981 \else
3982   \DeclareTextFontCommand{\textlatin}{\latintext}
3983 \fi

```

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

```

3984 \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.

```

3985 \bbl@trace{Loading basic (internal) bidi support}
3986 \ifodd\bbl@engine
3987 \else % TODO. Move to txtbabel
3988   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200 % Any xe+lua bidi=
3989     \bbl@error
3990     {The bidi method 'basic' is available only in\\%
3991     luatex. I'll continue with 'bidi=default', so\\%
3992     expect wrong results}%
3993     {See the manual for further details.}%
3994     \let\bbl@beforeforeign\leavevmode
3995     \AtEndOfPackage{%
3996       \EnableBabelHook{babel-bidi}%
3997       \bbl@xebidipar}
3998   \fi\fi
3999   \def\bbl@loadxebidi#1{%
4000     \ifx\RTLfootnotetext\@undefined
4001       \AtEndOfPackage{%

```

```

4002     \EnableBabelHook{babel-bidi}%
4003     \bbl@loadfontspec % bidi needs fontspec
4004     \usepackage#1{bidi}%
4005     \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
4006     \def\DigitsDotDashInterCharToks{% See the 'bidi' package
4007         \ifnum\@nameuse{bbl@wdir@\languagename}=\tw@ % 'AL' bidi
4008             \bbl@digitsdotdash % So ignore in 'R' bidi
4009         \fi}%
4010     \fi}
4011 \ifnum\bbl@bidimode>200 % Any xe bidi=
4012     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
4013         \bbl@tentative{bidi=bidi}
4014         \bbl@loadxebidi{}
4015     \or
4016         \bbl@loadxebidi{[rldocument]}
4017     \or
4018         \bbl@loadxebidi{}
4019     \fi
4020 \fi
4021 \fi
4022 % TODO? Separate:
4023 \ifnum\bbl@bidimode=\@ne % Any bidi= except default=1
4024     \let\bbl@beforeforeign\leavevmode
4025     \ifodd\bbl@engine
4026         \newattribute\bbl@attr@dir
4027         \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
4028         \bbl@exp{\output{\bodydir\pagedir\the\output}}
4029     \fi
4030     \AtEndOfPackage{%
4031         \EnableBabelHook{babel-bidi}%
4032         \ifodd\bbl@engine\else
4033             \bbl@xebidipar
4034         \fi}
4035 \fi

```

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

```

4036 \bbl@trace{Macros to switch the text direction}
4037 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
4038 \def\bbl@rscripts{% TODO. Base on codes ??
4039     ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
4040     Old Hungarian,Lydian,Mandaean,Manichaeen,%
4041     Meroitic Cursive,Meroitic,Old North Arabian,%
4042     Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
4043     Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
4044     Old South Arabian,}%
4045 \def\bbl@provide@dirs#1{%
4046     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
4047     \ifin@
4048         \global\bbl@csarg\chardef{wdir@#1}\@ne
4049         \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4050     \ifin@
4051         \global\bbl@csarg\chardef{wdir@#1}\tw@
4052     \fi
4053     \else
4054         \global\bbl@csarg\chardef{wdir@#1}\z@
4055     \fi
4056     \ifodd\bbl@engine
4057         \bbl@csarg\ifcase{wdir@#1}%
4058             \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
4059         \or
4060             \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
4061         \or

```



```

4062 \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
4063 \fi
4064 \fi}
4065 \def\bb@switchdir{%
4066 \bb@ifunset{\bb@lsys@\language name}{\bb@provide@lsys{\language name}}{}%
4067 \bb@ifunset{\bb@wdir@\language name}{\bb@provide@dirs{\language name}}{}%
4068 \bb@exp{\bb@setdirs\bb@cl{wdir}}
4069 \def\bb@setdirs#1{% TODO - math
4070 \ifcase\bb@select@type % TODO - strictly, not the right test
4071 \bb@bodydir{#1}%
4072 \bb@pdir{#1}% <- Must precede \bb@textdir
4073 \fi
4074 \bb@textdir{#1}}
4075 % TODO. Only if \bb@bidimode > 0?:
4076 \AddBabelHook{babel-bidi}{afterextras}{\bb@switchdir}
4077 \DisableBabelHook{babel-bidi}

```

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

```

4078 \ifodd\bb@engine % luatex=1
4079 \else % pdftex=0, xetex=2
4080 \newcount\bb@dirlevel
4081 \chardef\bb@thetextdir\z@
4082 \chardef\bb@thepardir\z@
4083 \def\bb@textdir#1{%
4084 \ifcase#1\relax
4085 \chardef\bb@thetextdir\z@
4086 \@nameuse{setlatin}%
4087 \bb@textdir@i\beginL\endL
4088 \else
4089 \chardef\bb@thetextdir\@ne
4090 \@nameuse{setnonlatin}%
4091 \bb@textdir@i\beginR\endR
4092 \fi}
4093 \def\bb@textdir@i#1#2{%
4094 \ifhmode
4095 \ifnum\currentgrouplevel>\z@
4096 \ifnum\currentgrouplevel=\bb@dirlevel
4097 \bb@error{Multiple bidi settings inside a group}%
4098 {I'll insert a new group, but expect wrong results.}%
4099 \bgroup\aftergroup#2\aftergroup\egroup
4100 \else
4101 \ifcase\currentgrouptype\or % 0 bottom
4102 \aftergroup#2% 1 simple {}
4103 \or
4104 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4105 \or
4106 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4107 \or\or\or % vbox vtop align
4108 \or
4109 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4110 \or\or\or\or\or\or % output math disc insert vcent mathchoice
4111 \or
4112 \aftergroup#2% 14 \begingroup
4113 \else
4114 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4115 \fi
4116 \fi
4117 \bb@dirlevel\currentgrouplevel
4118 \fi
4119 #1%
4120 \fi}
4121 \def\bb@pdir#1{\chardef\bb@thepardir#1\relax}
4122 \let\bb@bodydir\@gobble

```

```

4123 \let\bbl@pagedir@gobble
4124 \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).

```

4125 \def\bbl@xebidipar{%
4126   \let\bbl@xebidipar\relax
4127   \TeXeTstate@one
4128   \def\bbl@xeeverypar{%
4129     \ifcase\bbl@thepardir
4130       \ifcase\bbl@thetextdir\else\beginR\fi
4131     \else
4132       {\setbox\z@\lastbox\beginR\box\z@}%
4133     \fi}%
4134   \let\bbl@severypar\everypar
4135   \newtoks\everypar
4136   \everypar=\bbl@severypar
4137   \bbl@severypar{\bbl@xeeverypar\the\everypar}}
4138 \ifnum\bbl@bidimode>200 % Any xe bidi=
4139   \let\bbl@textdir@i@gobbletwo
4140   \let\bbl@xebidipar@empty
4141   \AddBabelHook{bidi}{foreign}{%
4142     \def\bbl@tempa{\def\BabelText###1}%
4143     \ifcase\bbl@thetextdir
4144       \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4145     \else
4146       \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4147     \fi}
4148   \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4149 \fi
4150 \fi

```

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

```

4151 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4152 \AtBeginDocument{%
4153   \ifx\pdfstringdefDisableCommands\@undefined\else
4154     \ifx\pdfstringdefDisableCommands\relax\else
4155       \pdfstringdefDisableCommands{\let\babelsublr@firstofone}%
4156     \fi
4157   \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`.

```

4158 \bbl@trace{Local Language Configuration}
4159 \ifx\loadlocalcfg@undefined
4160   \@ifpackagewith{babel}{noconfigs}%
4161     {\let\loadlocalcfg@gobble}%
4162     {\def\loadlocalcfg#1{%
4163       \InputIfFileExists{#1.cfg}%
4164       {\typeout{*****^J%
4165                * Local config file #1.cfg used^^J%
4166                *}}}%
4167     \@empty}}
4168 \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).

```
4169 \bbl@trace{Language options}
4170 \let\bbl@afterlang\relax
4171 \let\BabelModifiers\relax
4172 \let\bbl@loaded\@empty
4173 \def\bbl@load@language#1{%
4174   \InputIfFileExists{#1.ldf}%
4175   {\edef\bbl@loaded{\CurrentOption
4176     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4177     \expandafter\let\expandafter\bbl@afterlang
4178     \csname\CurrentOption.ldf-h@k\endcsname
4179     \expandafter\let\expandafter\BabelModifiers
4180     \csname bbl@mod@\CurrentOption\endcsname
4181     \bbl@exp{\AtBeginDocument{%
4182       \bbl@usehooks@lang{\CurrentOption}{begindocument}{\CurrentOption}}}}%
4183   {\bbl@error{%
4184     Unknown option '\CurrentOption'. Either you misspelled it\%
4185     or the language definition file \CurrentOption.ldf was not found}%
4186     Valid options are, among others: shorthands=, KeepShorthandsActive,\%
4187     activeacute, activegrave, noconfigs, safe=, main=, math=\%
4188     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.

```
4189 \def\bbl@try@load@lang#1#2#3{%
4190   \IfFileExists{\CurrentOption.ldf}%
4191   {\bbl@load@language{\CurrentOption}}%
4192   {#1\bbl@load@language{#2#3}}
4193 %
4194 \DeclareOption{hebrew}{%
4195   \input{rlbabel.def}%
4196   \bbl@load@language{hebrew}}
4197 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4198 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4199 \DeclareOption{northern sami}{\bbl@try@load@lang{}{samin}{}}
4200 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4201 \DeclareOption{polutonikogreek}{%
4202   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4203 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4204 \DeclareOption{scottishgaelic}{\bbl@try@load@lang{}{scottish}{}}
4205 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4206 \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.

```
4207 \ifx\bbl@opt@config\@nnil
4208   \@ifpackagewith{babel}{noconfigs}{}%
4209   {\InputIfFileExists{bblopts.cfg}%
4210     {\typeout{*****^J%
4211       * Local config file bblopts.cfg used^^J%
4212       *}}%
4213     {}}%
4214 \else
4215   \InputIfFileExists{\bbl@opt@config.cfg}%
4216   {\typeout{*****^J%
4217     * Local config file \bbl@opt@config.cfg used^^J%
4218     *}}%
```

```

4219   {\bbl@error{%
4220     Local config file '\bbl@opt@config.cfg' not found}{%
4221     Perhaps you misspelled it.}}%
4222 \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.

```

4223 \ifx\bbl@opt@main\@nnil
4224   \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4225     \let\bbl@tempb\@empty
4226     \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4227     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4228     \bbl@foreach\bbl@tempb{%   \bbl@tempb is a reversed list
4229       \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4230         \ifodd\bbl@iniflag % = *=
4231           \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
4232         \else % n +=
4233           \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4234         \fi
4235       \fi}%
4236   \fi
4237 \else
4238   \bbl@info{Main language set with 'main='. Except if you have\\%
4239     problems, prefer the default mechanism for setting\\%
4240     the main language, ie, as the last declared.\\%
4241     Reported}
4242 \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`).

```

4243 \ifx\bbl@opt@main\@nnil\else
4244   \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
4245   \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4246 \fi

```

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

```

4247 \bbl@foreach\bbl@language@opts{%
4248   \def\bbl@tempa{#1}%
4249   \ifx\bbl@tempa\bbl@opt@main\else
4250     \ifnum\bbl@iniflag<\tw@ % 0 ∅ (other = ldf)
4251       \bbl@ifunset{ds@#1}%
4252       {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4253     {}%
4254   \else % + * (other = ini)
4255     \DeclareOption{#1}{%
4256       \bbl@ldfinit
4257       \babelprovide[import]{#1}%
4258       \bbl@afterldf{}}%
4259   \fi
4260 \fi}
4261 \bbl@foreach\@classoptionslist{%
4262   \def\bbl@tempa{#1}%
4263   \ifx\bbl@tempa\bbl@opt@main\else
4264     \ifnum\bbl@iniflag<\tw@ % 0 ∅ (other = ldf)
4265       \bbl@ifunset{ds@#1}%
4266       {\IfFileExists{#1.ldf}%
4267        {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4268       {}}%

```

```

4269     }%
4270     \else % + * (other = ini)
4271     \IfFileExists{babel-#1.tex}%
4272     {\DeclareOption{#1}{%
4273       \bbl@ldfinit
4274       \babelprovide[import]{#1}%
4275       \bbl@afterldf{}}}%
4276     }%
4277     \fi
4278     \fi}

```

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

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

```

4279 \def\AfterBabelLanguage#1{%
4280   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang{}}
4281 \DeclareOption*{}
4282 \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.

```

4283 \bbl@trace{Option 'main'}
4284 \ifx\bbl@opt@main\@nnil
4285   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4286   \let\bbl@tempc\@empty
4287   \edef\bbl@templ{,\bbl@loaded,}
4288   \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4289   \bbl@for\bbl@tempb\bbl@tempa{%
4290     \edef\bbl@tempd{,\bbl@tempb,}%
4291     \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4292     \bbl@xin@{\bbl@tempd}{\bbl@templ}%
4293     \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
4294   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4295   \expandafter\bbl@tempa\bbl@loaded,\@nnil
4296   \ifx\bbl@tempb\bbl@tempc\else
4297     \bbl@warning{%
4298       Last declared language option is '\bbl@tempc',\%
4299       but the last processed one was '\bbl@tempb'.\%
4300       The main language can't be set as both a global\%
4301       and a package option. Use 'main=\bbl@tempc' as\%
4302       option. Reported}
4303   \fi
4304 \else
4305   \ifodd\bbl@iniflag % case 1,3 (main is ini)
4306     \bbl@ldfinit
4307     \let\CurrentOption\bbl@opt@main
4308     \bbl@exp{% \bbl@opt@provide = empty if *
4309       \\\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4310     \bbl@afterldf{}
4311     \DeclareOption{\bbl@opt@main}{}
4312   \else % case 0,2 (main is ldf)
4313     \ifx\bbl@loadmain\relax
4314       \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4315     \else
4316       \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4317     \fi
4318     \ExecuteOptions{\bbl@opt@main}
4319     \@namedef{ds@\bbl@opt@main}{}%
4320   \fi

```

```

4321 \DeclareOption*{}
4322 \ProcessOptions*
4323 \fi
4324 \bbl@exp{%
4325   \\AtBeginDocument{\\bbl@usehooks@lang{/}{begindocument}{}}}%
4326 \def\AfterBabelLanguage{%
4327   \bbl@error
4328   {Too late for \string\AfterBabelLanguage}%
4329   {Languages have been loaded, so I can do nothing}}

```

In order to catch the case where the user didn't specify a language we check whether `\bbl@main@language`, has become defined. If not, the `nil` language is loaded.

```

4330 \ifx\bbl@main@language\undefined
4331   \bbl@info{%
4332     You haven't specified a language as a class or package\\%
4333     option. I'll load 'nil'. Reported}
4334   \bbl@load@language{nil}
4335 \fi
4336 \</package>

```

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

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

Because plain \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`

```

4337 <*kernel>
4338 \let\bbl@onlyswitch\@empty
4339 \input babel.def
4340 \let\bbl@onlyswitch\@undefined
4341 </kernel>
4342 <*patterns>

```

7 Loading hyphenation patterns

The following code is meant to be read by `ini \TeX` 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.

```

4343 <<Make sure ProvidesFile is defined>>
4344 \ProvidesFile{hyphen.cfg}[<<date>> v<<version>> Babel hyphens]
4345 \xdef\bbl@format{\jobname}
4346 \def\bbl@version{<<version>>}
4347 \def\bbl@date{<<date>>}
4348 \ifx\AtBeginDocument\undefined
4349   \def\@empty{}
4350 \fi
4351 <<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.

```

4352 \def\process@line#1#2 #3 #4 {%
4353   \ifx=#1%
4354     \process@synonym{#2}%

```

```

4355 \else
4356   \process@language{#1#2}{#3}{#4}%
4357 \fi
4358 \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.

```

4359 \toks@{}
4360 \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.

```

4361 \def\process@synonym#1{%
4362   \ifnum\last@language=\m@ne
4363     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4364   \else
4365     \expandafter\chardef\csname l@#1\endcsname\last@language
4366     \wlog{\string\l@#1=\string\language\the\last@language}%
4367     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4368       \csname\language\hyphenmins\endcsname
4369     \let\bbl@elt\relax
4370     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}{}%
4371   \fi}

```

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

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

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

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

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

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

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

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

`\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.

```

4372 \def\process@language#1#2#3{%
4373   \expandafter\addlanguage\csname l@#1\endcsname
4374   \expandafter\language\csname l@#1\endcsname
4375   \edef\language#1%
4376   \bbl@hook@everylanguage{#1}%
4377   % > luatex
4378   \bbl@get@enc#1: :@@@
4379   \begingroup
4380   \lefthyphenmin\m@ne

```

```

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

```

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

```
4407 \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.

```

4408 \def\bbl@hook@everylanguage#1{}
4409 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4410 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4411 \def\bbl@hook@loadkernel#1{%
4412 \def\addlanguage{\csname newlanguage\endcsname}%
4413 \def\adddialect##1##2{%
4414 \global\chardef##1##2\relax
4415 \wlog{\string##1 = a dialect from \string\language##2}}%
4416 \def\iflanguage##1{%
4417 \expandafter\ifx\csname l@##1\endcsname\relax
4418 \nol@nerr{##1}%
4419 \else
4420 \ifnum\csname l@##1\endcsname=\language
4421 \expandafter\expandafter\expandafter\@firstoftwo
4422 \else
4423 \expandafter\expandafter\expandafter\@secondoftwo
4424 \fi
4425 \fi}%
4426 \def\providehyphenmins##1##2{%
4427 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4428 \@namedef{##1hyphenmins}{##2}%
4429 \fi}%
4430 \def\set@hyphenmins##1##2{%
4431 \lefthyphenmin##1\relax
4432 \righthyphenmin##2\relax}%
4433 \def\selectlanguage{%
4434 \errhelp{Selecting a language requires a package supporting it}%
4435 \errmessage{Not loaded}}%
4436 \let\foreignlanguage\selectlanguage

```



```

4437 \let\otherlanguage\selectlanguage
4438 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4439 \def\bb1@usehooks##1##2{% TODO. Temporary!!
4440 \def\setlocale{%
4441   \errhelp{Find an armchair, sit down and wait}%
4442   \errmessage{Not yet available}}%
4443 \let\uselocale\setlocale
4444 \let\locale\setlocale
4445 \let\selectlocale\setlocale
4446 \let\localename\setlocale
4447 \let\textlocale\setlocale
4448 \let\textlanguage\setlocale
4449 \let\languagegettext\setlocale}
4450 \begingroup
4451 \def\AddBabelHook#1#2{%
4452   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4453   \def\next{\toks1}%
4454   \else
4455     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4456   \fi
4457   \next}
4458 \ifx\directlua@undefined
4459   \ifx\XeTeXinputencoding@undefined\else
4460     \input xebabel.def
4461   \fi
4462 \else
4463   \input luababel.def
4464 \fi
4465 \openin1 = babel-\bbl@format.cfg
4466 \ifeof1
4467 \else
4468   \input babel-\bbl@format.cfg\relax
4469 \fi
4470 \closein1
4471 \endgroup
4472 \bbl@hook@loadkernel{switch.def}

```

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

```

4473 \openin1 = language.dat

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

4474 \def\language{english}%
4475 \ifeof1
4476 \message{I couldn't find the file language.dat,\space
4477           I will try the file hyphen.tex}
4478 \input hyphen.tex\relax
4479 \chardef\l@english\z@
4480 \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`.

```
4481 \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.

```

4482 \loop
4483   \endlinechar\m@ne
4484   \read1 to \bbl@line
4485   \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.

```
4486 \if T\ifeof1F\fi T\relax
4487 \ifx\bbl@line\@empty\else
4488 \edef\bbl@line{\bbl@line\space\space\space}%
4489 \expandafter\process@line\bbl@line\relax
4490 \fi
4491 \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.

```
4492 \begingroup
4493 \def\bbl@elt#1#2#3#4{%
4494 \global\language=#2\relax
4495 \gdef\languagename{#1}%
4496 \def\bbl@elt##1##2##3##4{}}%
4497 \bbl@languages
4498 \endgroup
4499 \fi
4500 \closein1
```

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

```
4501 \if/\the\toks@\else
4502 \errhelp{language.dat loads no language, only synonyms}
4503 \errmessage{Orphan language synonym}
4504 \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.

```
4505 \let\bbl@line\@undefined
4506 \let\process@line\@undefined
4507 \let\process@synonym\@undefined
4508 \let\process@language\@undefined
4509 \let\bbl@get@enc\@undefined
4510 \let\bbl@hyph@enc\@undefined
4511 \let\bbl@tempa\@undefined
4512 \let\bbl@hook@loadkernel\@undefined
4513 \let\bbl@hook@everylanguage\@undefined
4514 \let\bbl@hook@loadpatterns\@undefined
4515 \let\bbl@hook@loadexceptions\@undefined
4516 </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].

```
4517 <<(*More package options)>> ≡
4518 \chardef\bbl@bidimode\z@
4519 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4520 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4521 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4522 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4523 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4524 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4525 <</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.

```

4526 <<*Font selection>> ≡
4527 \bbl@trace{Font handling with fontspec}
4528 \ifx\ExplSyntax0n\@undefined\else
4529   \def\bbl@fs@warn@nx#1#2{% \bbl@tempfs is the original macro
4530     \in@{,#1,}{,no-script,language-not-exist,}%
4531     \ifin@ \else\bbl@tempfs@nx{#1}{#2}\fi}
4532   \def\bbl@fs@warn@nxx#1#2#3{%
4533     \in@{,#1,}{,no-script,language-not-exist,}%
4534     \ifin@ \else\bbl@tempfs@nxx{#1}{#2}{#3}\fi}
4535   \def\bbl@loadfontspec{%
4536     \let\bbl@loadfontspec\relax
4537     \ifx\fontspec\@undefined
4538       \usepackage{fontspec}%
4539     \fi}%
4540 \fi
4541 \@onlypreamble\babelfont
4542 \newcommand\babelfont[2][]{% 1=langs/scripts 2=fam
4543   \bbl@foreach{#1}{%
4544     \expandafter\ifx\csname date##1\endcsname\relax
4545       \IfFileExists{babel-##1.tex}%
4546         {\babelprovide{##1}}%
4547       {}%
4548     \fi}%
4549   \edef\bbl@tempa{#1}%
4550   \def\bbl@tempb{#2}% Used by \bbl@bblfont
4551   \bbl@loadfontspec
4552   \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4553   \bbl@bblfont}
4554 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
4555   \bbl@ifunset{\bbl@tempb family}%
4556   {\bbl@providefam{\bbl@tempb}}%
4557   {}%
4558   % For the default font, just in case:
4559   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}}%
4560   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4561   {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
4562   \bbl@exp{%
4563     \let<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
4564     \\\bbl@font@set<bbl@\bbl@tempb dflt@\languagename>%
4565     \<bbl@tempb default>\<bbl@tempb family>}}%
4566   {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4567     \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:

```

4568 \def\bbl@providefam#1{%
4569   \bbl@exp{%
4570     \\\newcommand<#1default>{}% Just define it
4571     \\\bbl@add@list\\bbl@font@fams{#1}%
4572     \\\DeclareRobustCommand<#1family>{%
4573       \\\not@math@alphabet<#1family>\relax
4574       % \\\prepare@family@series@update{#1}<#1default>% TODO. Fails
4575       \\\fontfamily<#1default>%
4576       \<ifx>\\UseHooks\\@undefined\<else>\\UseHook{#1family}\<fi>%
4577       \\\selectfont}%
4578     \\\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.

```

4579 \def\bbl@nostdfont#1{%
4580   \bbl@ifunset{bbl@WFF@\f@family}%

```

```

4581 {\bbl@csarg\gdef{WFF@f@family}}}% Flag, to avoid dupl warns
4582 \bbl@infowarn{The current font is not a babel standard family:\%
4583 #1%
4584 \fontname\font\%
4585 There is nothing intrinsically wrong with this warning, and\%
4586 you can ignore it altogether if you do not need these\%
4587 families. But if they are used in the document, you should be\%
4588 aware 'babel' will not set Script and Language for them, so\%
4589 you may consider defining a new family with \string\babelfont.\%
4590 See the manual for further details about \string\babelfont.\%
4591 Reported}}
4592 {}}%
4593 \gdef\bbl@switchfont{%
4594 \bbl@ifunset{\bbl@lsys@\language}\bbl@provide@lsys{\language}}}%
4595 \bbl@exp{% eg Arabic -> arabic
4596 \lowercase{\edef{\bbl@tempa{\bbl@cl{sname}}}}}%
4597 \bbl@foreach\bbl@font@fams{%
4598 \bbl@ifunset{\bbl@##1dflt@\language}% (1) language?
4599 {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}% (2) from script?
4600 {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
4601 }% 123=F - nothing!
4602 {\bbl@exp{% 3=T - from generic
4603 \global\let\bbl@##1dflt@\language}%
4604 \<bbl@##1dflt@>}}}%
4605 {\bbl@exp{% 2=T - from script
4606 \global\let\bbl@##1dflt@\language}%
4607 \<bbl@##1dflt@*\bbl@tempa>}}}%
4608 {}}% 1=T - language, already defined
4609 \def\bbl@tempa{\bbl@nostdfont}}}% TODO. Don't use \bbl@tempa
4610 \bbl@foreach\bbl@font@fams{% don't gather with prev for
4611 \bbl@ifunset{\bbl@##1dflt@\language}%
4612 {\bbl@cs{famrst@##1}%
4613 \global\bbl@csarg\let{famrst@##1}\relax}%
4614 {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4615 \bbl@add{\originalTeX}%
4616 \bbl@font@rst{\bbl@cl{##1dflt}}}%
4617 \<##1default>\<##1family>{##1}}}%
4618 \bbl@font@set{\bbl@##1dflt@\language}% the main part!
4619 \<##1default>\<##1family>}}}%
4620 \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.

```

4621 \ifx\f@family\undefined\else % if latex
4622 \ifcase\bbl@engine % if pdftex
4623 \let\bbl@cckckstdfonts\relax
4624 \else
4625 \def\bbl@cckckstdfonts{%
4626 \begingroup
4627 \global\let\bbl@cckckstdfonts\relax
4628 \let\bbl@tempa\empty
4629 \bbl@foreach\bbl@font@fams{%
4630 \bbl@ifunset{\bbl@##1dflt@}%
4631 {\@nameuse{##1family}}%
4632 \bbl@csarg\gdef{WFF@f@family}}}% Flag
4633 \bbl@exp{\bbl@add{\bbl@tempa{* \<##1family>= \f@family\%
4634 \space\space\fontname\font\%
4635 \bbl@csarg\xdef{##1dflt@}{\f@family}}%
4636 \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4637 {}}%
4638 \ifx\bbl@tempa\empty\else
4639 \bbl@infowarn{The following font families will use the default\%
4640 settings for all or some languages:\%

```

```

4641         \bbl@tempa
4642         There is nothing intrinsically wrong with it, but\\%
4643         'babel' will no set Script and Language, which could\\%
4644         be relevant in some languages. If your document uses\\%
4645         these families, consider redefining them with \string\babelfont.\\%
4646         Reported}%
4647         \fi
4648     \endgroup}
4649 \fi
4650 \fi

```

Now the macros defining the font with fontspec.

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

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

```

4651 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4652   \bbl@xin@{<>}{#1}%
4653   \ifin@
4654     \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4655     \fi
4656     \bbl@exp{%
4657       \def\\#2{#1}%           'Unprotected' macros return prev values
4658       \\bbl@ifsamestring{#2}{\f@family}%
4659       {\#3%
4660         \\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}{}%
4661         \let\\bbl@tempa\relax}%
4662       {}}
4663 %   TODO - next should be global?, but even local does its job. I'm
4664 %   still not sure -- must investigate:
4665 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4666   \let\bbl@tempa\bbl@mapselect
4667   \edef\bbl@tempb{\bbl@stripslash#4/}% Catcodes hack (better pass it).
4668   \bbl@exp{\\bbl@replace\\bbl@tempb{\bbl@stripslash\family/}{}}%
4669   \let\bbl@mapselect\relax
4670   \let\bbl@tempa@fam#4%       eg, '\rmfamily', to be restored below
4671   \let#4@empty %             Make sure \renewfontfamily is valid
4672   \bbl@exp{%
4673     \let\\bbl@tempa@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
4674     \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4675     {\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4676     \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4677     {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4678     \let\\bbl@tempfs@nx<__fontspec_warning:nx>%
4679     \let<__fontspec_warning:nx>\\bbl@fs@warn@nx
4680     \let\\bbl@tempfs@nxx<__fontspec_warning:nxx>%
4681     \let<__fontspec_warning:nxx>\\bbl@fs@warn@nxx
4682     \\renewfontfamily\\#4%
4683     [\bbl@cl{lsys},%
4684     \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
4685     #2]}{#3}% ie \bbl@exp{..}{#3}
4686   \bbl@exp{%
4687     \let<__fontspec_warning:nx>\\bbl@tempfs@nx
4688     \let<__fontspec_warning:nxx>\\bbl@tempfs@nxx}%
4689   \begingroup
4690     #4%
4691     \xdef#1{\f@family}%       eg, \bbl@rmdflt@lang{FreeSerif(0)}
4692   \endgroup % TODO. Find better tests:

```

```

4693 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4694   {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
4695 \ifin@
4696   \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
4697 \fi
4698 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4699   {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
4700 \ifin@
4701   \global\bbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
4702 \fi
4703 \let#4\bbl@temp@fam
4704 \bbl@exp{\let<\bbl@stripslash#4\space>}\bbl@temp@pfam
4705 \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.

```

4706 \def\bbl@font@rst#1#2#3#4{%
4707   \bbl@csarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}

```

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

```

4708 \def\bbl@font@fams{rm, sf, tt}
4709 <</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.

```

4710 <<{*Footnote changes}>> ≡
4711 \bbl@trace{Bidi footnotes}
4712 \ifnum\bbl@bidimode>\z@ % Any bidi=
4713   \def\bbl@footnote#1#2#3{%
4714     \@ifnextchar[%
4715       {\bbl@footnote@o{#1}{#2}{#3}}%
4716       {\bbl@footnote@x{#1}{#2}{#3}}}%
4717   \long\def\bbl@footnote@x#1#2#3#4{%
4718     \bgroup
4719     \select@language@x{\bbl@main@language}%
4720     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4721     \egroup}
4722   \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4723     \bgroup
4724     \select@language@x{\bbl@main@language}%
4725     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4726     \egroup}
4727   \def\bbl@footnotetext#1#2#3{%
4728     \@ifnextchar[%
4729       {\bbl@footnotetext@o{#1}{#2}{#3}}%
4730       {\bbl@footnotetext@x{#1}{#2}{#3}}}%
4731   \long\def\bbl@footnotetext@x#1#2#3#4{%
4732     \bgroup
4733     \select@language@x{\bbl@main@language}%
4734     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4735     \egroup}
4736   \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4737     \bgroup
4738     \select@language@x{\bbl@main@language}%
4739     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4740     \egroup}
4741   \def\BabelFootnote#1#2#3#4{%
4742     \ifx\bbl@fn@footnote\undefined

```

```

4743 \let\bbbl@fn@footnote\footnote
4744 \fi
4745 \ifx\bbbl@fn@footnotetext\undefined
4746 \let\bbbl@fn@footnotetext\footnotetext
4747 \fi
4748 \bbl@ifblank{#2}%
4749 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4750 \@namedef{\bbl@stripslash#1text}%
4751 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4752 {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}}%
4753 \@namedef{\bbl@stripslash#1text}%
4754 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}}
4755 \fi
4756 <</Footnote changes>>

```

Now, the code.

```

4757 (*xetex)
4758 \def\BabelStringsDefault{unicode}
4759 \let\xebbl@stop\relax
4760 \AddBabelHook{xetex}{encodedcommands}{%
4761 \def\bbbl@tempa{#1}%
4762 \ifx\bbbl@tempa\empty
4763 \XeTeXinputencoding"bytes"%
4764 \else
4765 \XeTeXinputencoding"#1"%
4766 \fi
4767 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4768 \AddBabelHook{xetex}{stopcommands}{%
4769 \xebbl@stop
4770 \let\xebbl@stop\relax}
4771 \def\bbbl@intraspace#1 #2 #3\@@{%
4772 \bbl@csarg\gdef{xeisp@\languagename}%
4773 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4774 \def\bbbl@intrapenalty#1\@@{%
4775 \bbl@csarg\gdef{xeipn@\languagename}%
4776 {\XeTeXlinebreakpenalty #1\relax}}
4777 \def\bbbl@provide@intraspace{%
4778 \bbl@xin@{/s}{/\bbl@cl{lbrk}}}%
4779 \ifin@else\bbl@xin@{/c}{/\bbl@cl{lbrk}}\fi
4780 \ifin@
4781 \bbl@ifunset{\bbl@intsp@\languagename}{}%
4782 {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\empty\else
4783 \ifx\bbl@KVP@intraspace\@nnil
4784 \bbl@exp{%
4785 \bbl@intraspace\bbl@cl{intsp}\@@}%
4786 \fi
4787 \ifx\bbl@KVP@intrapenalty\@nnil
4788 \bbl@intrapenalty0\@@
4789 \fi
4790 \fi
4791 \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4792 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4793 \fi
4794 \ifx\bbl@KVP@intrapenalty\@nnil\else
4795 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4796 \fi
4797 \bbl@exp{%
4798 % TODO. Execute only once (but redundant):
4799 \bbl@add\<extras\languagename>%
4800 \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
4801 \<bbl@xeisp@\languagename>%
4802 \<bbl@xeipn@\languagename>%
4803 \bbl@toglobal\<extras\languagename>%

```

```

4804     \\bbl@add\<noextras\languagename>{%
4805         \XeTeXlinebreaklocale ""}%
4806     \\bbl@toglobal\<noextras\languagename>}%
4807     \ifx\bbl@ispace\undefined
4808         \gdef\bbl@ispace{\bbl@cl{xisp}}%
4809     \ifx\AtBeginDocument\@notprerr
4810         \expandafter\@secondoftwo % to execute right now
4811         \fi
4812     \AtBeginDocument{\bbl@patchfont{\bbl@ispace}}%
4813     \fi}%
4814 \fi}
4815 \ifx\DisableBabelHook\undefined\endinput\fi
4816 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4817 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4818 \DisableBabelHook{babel-fontspec}
4819 <<Font selection>>
4820 \def\bbl@provide@extra#1{}
4821 </xetex>

```

9.2 Layout

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

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

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

```

4822 (*xetex | texxet)
4823 \providecommand\bbl@provide@intraspace{}
4824 \bbl@trace{Redefinitions for bidi layout}
4825 \def\bbl@sspre@caption{%
4826     \bbl@exp{\everyhbox{\\\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4827 \ifx\bbl@opt@layout\@nnil\else % if layout=..
4828 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4829 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4830 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4831     \def\@hangfrom#1{%
4832         \setbox\@tempboxa\hbox{#1}}%
4833         \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4834         \noindent\box\@tempboxa}
4835     \def\raggedright{%
4836         \let\\\@centercr
4837         \bbl@startskip\z@skip
4838         \@rightskip\@flushglue
4839         \bbl@endskip\@rightskip
4840         \parindent\z@
4841         \parfillskip\bbl@startskip}
4842     \def\raggedleft{%
4843         \let\\\@centercr
4844         \bbl@startskip\@flushglue
4845         \bbl@endskip\z@skip
4846         \parindent\z@
4847         \parfillskip\bbl@endskip}
4848 \fi
4849 \IfBabelLayout{lists}
4850     {\bbl@sreplace\list
4851         {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4852     \def\bbl@listleftmargin{%
4853         \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4854     \ifcase\bbl@engine
4855         \def\labelenumii{}\theenumii{}% pdftex doesn't reverse ()
4856         \def\p@enumiii{\p@enumii}\theenumii{}%
4857     \fi

```



```

4858 \bbl@sreplace\@verbatim
4859   {\leftskip\@totalleftmargin}%
4860   {\bbl@startskip\textwidth
4861    \advance\bbl@startskip-\linewidth}%
4862 \bbl@sreplace\@verbatim
4863   {\rightskip\z@skip}%
4864   {\bbl@endskip\z@skip}}%
4865 {}
4866 \IfBabelLayout{contents}
4867   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4868    \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4869   {}
4870 \IfBabelLayout{columns}
4871   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4872    \def\bbl@outputbox#1{%
4873     \hb@xt@\textwidth{%
4874      \hskip\columnwidth
4875      \hfil
4876      {\normalcolor\vrule \@width\columnseprule}%
4877      \hfil
4878      \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4879      \hskip-\textwidth
4880      \hb@xt@\columnwidth{\box\@outputbox \hss}%
4881      \hskip\columnsep
4882      \hskip\columnwidth}}}%
4883   {}
4884 \langle Footnote changes \rangle
4885 \IfBabelLayout{footnotes}%
4886   {\BabelFootnote\footnote\languagename}{}%
4887   \BabelFootnote\localfootnote\languagename}{}%
4888   \BabelFootnote\mainfootnote}{%}
4889   {}

```

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.

```

4890 \IfBabelLayout{counters*}%
4891   {\bbl@add\bbl@opt@layout{.counters.}%
4892    \AddToHook{shipout/before}{%
4893     \let\bbl@tempa\babelsublr
4894     \let\babelsublr\@firstofone
4895     \let\bbl@save@thepage\thepage
4896     \protected@edef\thepage{\thepage}%
4897     \let\babelsublr\bbl@tempa}%
4898    \AddToHook{shipout/after}{%
4899     \let\thepage\bbl@save@thepage}}%
4900 \IfBabelLayout{counters}%
4901   {\let\bbl@latinarabic=\@arabic
4902    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4903    \let\bbl@asciroman=\@roman
4904    \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4905    \let\bbl@asciiRoman=\@Roman
4906    \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}%
4907 \fi % end if layout
4908 \</xetex | texxtet)

```

9.3 8-bit TeX

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

```

4909 (*texxtet)
4910 \def\bbl@provide@extra#1{%
4911   % == auto-select encoding ==
4912   \ifx\bbl@encoding@select@off\@empty\else
4913     \bbl@ifunset{bbl@encoding@#1}%

```

```

4914     {\def\@elt##1{,##1,}%
4915     \edef\bb@l@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
4916     \count@\z@
4917     \bb@l@foreach\bb@l@tempe{%
4918         \def\bb@l@tempd{##1}% Save last declared
4919         \advance\count@\@ne}%
4920     \ifnum\count@>\@ne
4921         \getlocaleproperty*\bb@l@tempa{#1}{identification/encodings}%
4922         \ifx\bb@l@tempa\relax \let\bb@l@tempa\@empty \fi
4923         \bb@l@replace\bb@l@tempa{ }{,}%
4924         \global\bb@l@csarg\let{encoding@#1}\@empty
4925         \bb@l@xin@{\bb@l@tempd,}{\bb@l@tempa,}%
4926         \ifin@\else % if main encoding included in ini, do nothing
4927             \let\bb@l@tempb\relax
4928             \bb@l@foreach\bb@l@tempa{%
4929                 \ifx\bb@l@tempb\relax
4930                     \bb@l@xin@{,##1,}{\bb@l@tempe,}%
4931                     \ifin@\def\bb@l@tempb{##1}\fi
4932                 \fi}%
4933             \ifx\bb@l@tempb\relax\else
4934                 \bb@l@exp{%
4935                     \global\<bb@l@add>\<bb@l@preextras@#1>{\<bb@l@encoding@#1>}%
4936                     \gdef\<bb@l@encoding@#1>{%
4937                         \\babel@save\\f@encoding
4938                         \\bb@l@add\\originalTeX{\\selectfont}%
4939                         \\fontencoding{\bb@l@tempb}%
4940                         \\selectfont}}%
4941                 \fi
4942             \fi
4943         \fi}%
4944     {}%
4945 \fi}
4946 /texet)

```

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, `\bb@l@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).

```

4947 (*luatex)
4948 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4949 \bbl@trace{Read language.dat}
4950 \ifx\bbl@readstream\@undefined
4951 \csname newread\endcsname\bbl@readstream
4952 \fi
4953 \beginingroup
4954 \toks@{}
4955 \count@z@ % 0=start, 1=0th, 2=normal
4956 \def\bbl@process@line#1#2 #3 #4 {%
4957 \ifx=#1%
4958 \bbl@process@synonym{#2}%
4959 \else
4960 \bbl@process@language{#1#2}{#3}{#4}%
4961 \fi
4962 \ignorespaces}
4963 \def\bbl@manylang{%
4964 \ifnum\bbl@last>\@ne
4965 \bbl@info{Non-standard hyphenation setup}%
4966 \fi
4967 \let\bbl@manylang\relax}
4968 \def\bbl@process@language#1#2#3{%
4969 \ifcase\count@
4970 \@ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
4971 \or
4972 \count@\tw@
4973 \fi
4974 \ifnum\count@=\tw@
4975 \expandafter\addlanguage\csname l@#1\endcsname
4976 \language\allocationnumber
4977 \chardef\bbl@last\allocationnumber
4978 \bbl@manylang
4979 \let\bbl@elt\relax
4980 \xdef\bbl@languages{%
4981 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4982 \fi
4983 \the\toks@
4984 \toks@{}}
4985 \def\bbl@process@synonym@aux#1#2{%
4986 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4987 \let\bbl@elt\relax
4988 \xdef\bbl@languages{%
4989 \bbl@languages\bbl@elt{#1}{#2}{}}}%
4990 \def\bbl@process@synonym#1{%
4991 \ifcase\count@
4992 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4993 \or
4994 \@ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}}%
4995 \else
4996 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4997 \fi}
4998 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4999 \chardef\l@english\z@
5000 \chardef\l@USenglish\z@
5001 \chardef\bbl@last\z@
5002 \global\@namedef{bbl@hyphendata@0}{\hyphen.tex}{}
5003 \gdef\bbl@languages{%
5004 \bbl@elt{english}{0}{hyphen.tex}}%

```

```

5005     \bbl@elt{USenglish}{0}{}}
5006 \else
5007     \global\let\bbl@languages@format\bbl@languages
5008     \def\bbl@elt#1#2#3#4{% Remove all except language 0
5009         \ifnum#2>\z@\else
5010             \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
5011         \fi}%
5012     \xdef\bbl@languages{\bbl@languages}%
5013 \fi
5014 \def\bbl@elt#1#2#3#4{\@namedef{zth#1}} % Define flags
5015 \bbl@languages
5016 \openin\bbl@readstream=language.dat
5017 \ifeof\bbl@readstream
5018     \bbl@warning{I couldn't find language.dat. No additional\\%
5019                 patterns loaded. Reported}%
5020 \else
5021     \loop
5022         \endlinechar\m@ne
5023         \read\bbl@readstream to \bbl@line
5024         \endlinechar`^^M
5025         \if T\ifeof\bbl@readstream F\fi T\relax
5026         \ifx\bbl@line\@empty\else
5027             \edef\bbl@line{\bbl@line\space\space\space}%
5028             \expandafter\bbl@process@line\bbl@line\relax
5029         \fi
5030     \repeat
5031 \fi
5032 \closein\bbl@readstream
5033 \endgroup
5034 \bbl@trace{Macros for reading patterns files}
5035 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc#2}}
5036 \ifx\babelcatcodetablenum\@undefined
5037     \ifx\newcatcodetable\@undefined
5038         \def\babelcatcodetablenum{5211}
5039         \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5040     \else
5041         \newcatcodetable\babelcatcodetablenum
5042         \newcatcodetable\bbl@pattcodes
5043     \fi
5044 \else
5045     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5046 \fi
5047 \def\bbl@luapatterns#1#2{%
5048     \bbl@get@enc#1:.\@@@
5049     \setbox\z@\hbox\bgroup
5050     \begingroup
5051         \savecatcodetable\babelcatcodetablenum\relax
5052         \initcatcodetable\bbl@pattcodes\relax
5053         \catcodetable\bbl@pattcodes\relax
5054         \catcode\#=6 \catcode\$\=3 \catcode\&\=4 \catcode\^=\7
5055         \catcode\_=\8 \catcode\{=\1 \catcode\}\=2 \catcode\~=\13
5056         \catcode\@=\11 \catcode\^^I=\10 \catcode\^^J=\12
5057         \catcode\<=\12 \catcode\>=\12 \catcode\*\=12 \catcode\.\=12
5058         \catcode\-\=12 \catcode\/\=12 \catcode\[=\12 \catcode\]=12
5059         \catcode\`=12 \catcode\`=12 \catcode\"=\12
5060         \input #1\relax
5061         \catcodetable\babelcatcodetablenum\relax
5062     \endgroup
5063     \def\bbl@tempa{#2}%
5064     \ifx\bbl@tempa\@empty\else
5065         \input #2\relax
5066     \fi
5067 \egroup}%

```

```

5068 \def\bbl@patterns@lua#1{%
5069   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5070     \csname l@#1\endcsname
5071     \edef\bbl@tempa{#1}%
5072   \else
5073     \csname l@#1:\f@encoding\endcsname
5074     \edef\bbl@tempa{#1:\f@encoding}%
5075   \fi\relax
5076   \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5077   \@ifundefined{bbl@hyphendata@the\language}%
5078     {\def\bbl@elt##1##2##3##4{%
5079       \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:0T1...
5080       \def\bbl@tempb{##3}%
5081       \ifx\bbl@tempb@empty\else % if not a synonymous
5082         \def\bbl@tempc{{##3}{##4}}%
5083       \fi
5084       \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5085     \fi}%
5086   \bbl@languages
5087   \@ifundefined{bbl@hyphendata@the\language}%
5088     {\bbl@info{No hyphenation patterns were set for\\%
5089       language '\bbl@tempa'. Reported}}%
5090     {\expandafter\expandafter\expandafter\bbl@luapatterns
5091       \csname bbl@hyphendata@the\language\endcsname}}}}
5092 \endinput\fi
5093 % Here ends \ifx\AddBabelHook\undefined
5094 % A few lines are only read by hyphen.cfg
5095 \ifx\DisableBabelHook\undefined
5096   \AddBabelHook{luatex}{everylanguage}{%
5097     \def\process@language##1##2##3{%
5098       \def\process@line####1####2 ####3 ####4 {}}}
5099   \AddBabelHook{luatex}{loadpatterns}{%
5100     \input #1\relax
5101     \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5102       {#{1}{}}}
5103   \AddBabelHook{luatex}{loadexceptions}{%
5104     \input #1\relax
5105     \def\bbl@tempb##1##2{{##1}{#1}}%
5106     \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5107       {\expandafter\expandafter\expandafter\bbl@tempb
5108         \csname bbl@hyphendata@the\language\endcsname}}
5109 \endinput\fi
5110 % Here stops reading code for hyphen.cfg
5111 % The following is read the 2nd time it's loaded
5112 \begingroup % TODO - to a lua file
5113 \catcode`\%=12
5114 \catcode`\'=12
5115 \catcode`\\"=12
5116 \catcode`\:=12
5117 \directlua{
5118   Babel = Babel or {}
5119   function Babel.bytes(line)
5120     return line:gsub(".",
5121       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5122   end
5123   function Babel.begin_process_input()
5124     if luatexbase and luatexbase.add_to_callback then
5125       luatexbase.add_to_callback('process_input_buffer',
5126         Babel.bytes, 'Babel.bytes')
5127     else
5128       Babel.callback = callback.find('process_input_buffer')
5129       callback.register('process_input_buffer', Babel.bytes)
5130     end

```

```

5131 end
5132 function Babel.end_process_input ()
5133   if luatexbase and luatexbase.remove_from_callback then
5134     luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5135   else
5136     callback.register('process_input_buffer', Babel.callback)
5137   end
5138 end
5139 function Babel.addpatterns(pp, lg)
5140   local lg = lang.new(lg)
5141   local pats = lang.patterns(lg) or ''
5142   lang.clear_patterns(lg)
5143   for p in pp:gmatch('[^%s]+') do
5144     ss = ''
5145     for i in string.utfcharacters(p:gsub('%d', '')) do
5146       ss = ss .. '%d?' .. i
5147     end
5148     ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
5149     ss = ss:gsub('%.%d%?$', '%%.')
5150     pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5151     if n == 0 then
5152       tex.sprint(
5153         [[\string\csname\space bbl@info\endcsname{New pattern: }
5154         .. p .. [{}]])
5155       pats = pats .. ' ' .. p
5156     else
5157       tex.sprint(
5158         [[\string\csname\space bbl@info\endcsname{Renew pattern: }
5159         .. p .. [{}]])
5160     end
5161   end
5162   lang.patterns(lg, pats)
5163 end
5164 Babel.characters = Babel.characters or {}
5165 Babel.ranges = Babel.ranges or {}
5166 function Babel.hlist_has_bidi(head)
5167   local has_bidi = false
5168   local ranges = Babel.ranges
5169   for item in node.traverse(head) do
5170     if item.id == node.id'glyph' then
5171       local itemchar = item.char
5172       local chardata = Babel.characters[itemchar]
5173       local dir = chardata and chardata.d or nil
5174       if not dir then
5175         for nn, et in ipairs(ranges) do
5176           if itemchar < et[1] then
5177             break
5178           elseif itemchar <= et[2] then
5179             dir = et[3]
5180             break
5181           end
5182         end
5183       end
5184       if dir and (dir == 'al' or dir == 'r') then
5185         has_bidi = true
5186       end
5187     end
5188   end
5189   return has_bidi
5190 end
5191 function Babel.set_chranges_b (script, chrng)
5192   if chrng == '' then return end
5193   texio.write('Replacing ' .. script .. ' script ranges')

```

```

5194 Babel.script_blocks[script] = {}
5195 for s, e in string.gmatch(chrng..' ', '(.)%.%.(-)%s') do
5196     table.insert(
5197         Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5198     end
5199 end
5200 function Babel.discard_sublr(str)
5201     if str:find( [[\string\indexentry]] ) and
5202         str:find( [[\string\babelsublr]] ) then
5203         str = str:gsub( [[\string\babelsublr%*(%b{})]],
5204             function(m) return m:sub(2,-2) end )
5205     end
5206     return str
5207 end
5208 }
5209 \endgroup
5210 \ifx\newattribute\undefined\else
5211     \newattribute\bbl@attr@locale
5212     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5213     \AddBabelHook{luatex}{beforeextras}{%
5214         \setattribute\bbl@attr@locale\localeid}
5215 \fi
5216 \def\BabelStringsDefault{unicode}
5217 \let\luabbl@stop\relax
5218 \AddBabelHook{luatex}{encodedcommands}{%
5219     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5220     \ifx\bbl@tempa\bbl@tempb\else
5221         \directlua{Babel.begin_process_input()}%
5222         \def\luabbl@stop{%
5223             \directlua{Babel.end_process_input()}%
5224         \fi}%
5225 \AddBabelHook{luatex}{stopcommands}{%
5226     \luabbl@stop
5227     \let\luabbl@stop\relax}
5228 \AddBabelHook{luatex}{patterns}{%
5229     \@ifundefined{bbl@hyphendata@the\language}%
5230     {\def\bbl@elt##1##2##3##4{%
5231         \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
5232         \def\bbl@tempb{##3}%
5233         \ifx\bbl@tempb\@empty\else % if not a synonymous
5234             \def\bbl@tempc{{##3}{##4}}%
5235         \fi
5236         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5237     \fi}%
5238     \bbl@languages
5239     \@ifundefined{bbl@hyphendata@the\language}%
5240     {\bbl@info{No hyphenation patterns were set for\%
5241         language '#2'. Reported}}%
5242     {\expandafter\expandafter\expandafter\bbl@luapatterns
5243         \csname bbl@hyphendata@the\language\endcsname}}}%
5244 \@ifundefined{bbl@patterns@}{%
5245     \begingroup
5246     \bbl@xin@{\, \number\language,}{, \bbl@pttnlist}%
5247     \ifin@ \else
5248         \ifx\bbl@patterns@\@empty\else
5249             \directlua{ Babel.addpatterns(
5250                 [[\bbl@patterns@]], \number\language) }%
5251         \fi
5252         \@ifundefined{bbl@patterns@#1}%
5253         \@empty
5254         {\directlua{ Babel.addpatterns(
5255             [[\space\csname bbl@patterns@#1\endcsname]],
5256             \number\language) }}%

```

```

5257     \xdef\bbbl@pttnlist{\bbbl@pttnlist\number\language,}%
5258     \fi
5259   \endgroup}%
5260 \bbbl@exp{%
5261   \bbbl@ifunset{\bbbl@prehc@\language\name}{}%
5262   {\bbbl@ifblank{\bbbl@cs{\prehc@\language\name}}{}}%
5263   {\prehyphenchar=\bbbl@c{\prehc}\relax}}}}

```

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

```

5264 \@onlypreamble\babelpatterns
5265 \AtEndOfPackage{%
5266   \newcommand\babelpatterns[2][\@empty]{%
5267     \ifx\bbbl@patterns@\relax
5268       \let\bbbl@patterns@\@empty
5269     \fi
5270     \ifx\bbbl@pttnlist\@empty\else
5271       \bbbl@warning{%
5272         You must not intermingle \string\selectlanguage\space and\%
5273         \string\babelpatterns\space or some patterns will not\%
5274         be taken into account. Reported}%
5275     \fi
5276     \ifx\@empty#1%
5277       \protected@edef\bbbl@patterns@{\bbbl@patterns@\space#2}%
5278     \else
5279       \edef\bbbl@tempb{\zap@space#1 \@empty}%
5280       \bbbl@for\bbbl@tempa\bbbl@tempb{%
5281         \bbbl@fixname\bbbl@tempa
5282         \bbbl@iflanguage\bbbl@tempa{%
5283           \bbbl@csarg\protected@edef{\patterns@\bbbl@tempa}{%
5284             \@ifundefined{\bbbl@patterns@\bbbl@tempa}%
5285             \@empty
5286             {\csname \bbbl@tempa\endcsname\space}%
5287             #2}}}%
5288     \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.

```

5289 % TODO - to a lua file
5290 \directlua{
5291   Babel = Babel or {}
5292   Babel.linebreaking = Babel.linebreaking or {}
5293   Babel.linebreaking.before = {}
5294   Babel.linebreaking.after = {}
5295   Babel.locale = {} % Free to use, indexed by \localeid
5296   function Babel.linebreaking.add_before(func, pos)
5297     tex.print([[noexpand\csname \bbbl@lua\hyphenate\endcsname]])
5298     if pos == nil then
5299       table.insert(Babel.linebreaking.before, func)
5300     else
5301       table.insert(Babel.linebreaking.before, pos, func)
5302     end
5303   end
5304   function Babel.linebreaking.add_after(func)
5305     tex.print([[noexpand\csname \bbbl@lua\hyphenate\endcsname]])
5306     table.insert(Babel.linebreaking.after, func)
5307   end

```



```

5308 }
5309 \def\bbl@intraspace#1 #2 #3\@@{%
5310   \directlua{
5311     Babel = Babel or {}
5312     Babel.intraspaces = Babel.intraspaces or {}
5313     Babel.intraspaces['\csname bbl@sbc@p@\languagename\endcsname'] = %
5314       {b = #1, p = #2, m = #3}
5315     Babel.locale_props[\the\localeid].intraspace = %
5316       {b = #1, p = #2, m = #3}
5317   }}
5318 \def\bbl@intrapenalty#1\@@{%
5319   \directlua{
5320     Babel = Babel or {}
5321     Babel.intrapenalties = Babel.intrapenalties or {}
5322     Babel.intrapenalties['\csname bbl@sbc@p@\languagename\endcsname'] = #1
5323     Babel.locale_props[\the\localeid].intrapenalty = #1
5324   }}
5325 \beginingroup
5326 \catcode`\%=12
5327 \catcode`\^=14
5328 \catcode`\'=12
5329 \catcode`\-=12
5330 \gdef\bbl@seaintraspace{^
5331   \let\bbl@seaintraspace\relax
5332   \directlua{
5333     Babel = Babel or {}
5334     Babel.sea_enabled = true
5335     Babel.sea_ranges = Babel.sea_ranges or {}
5336     function Babel.set_chrngs (script, chrng)
5337       local c = 0
5338       for s, e in string.gmatch(chrng..' ', '(.)%.%.(-)%s') do
5339         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5340         c = c + 1
5341       end
5342     end
5343     function Babel.sea_disc_to_space (head)
5344       local sea_ranges = Babel.sea_ranges
5345       local last_char = nil
5346       local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5347       for item in node.traverse(head) do
5348         local i = item.id
5349         if i == node.id'glyph' then
5350           last_char = item
5351         elseif i == 7 and item.subtype == 3 and last_char
5352           and last_char.char > 0x0C99 then
5353           quad = font.getfont(last_char.font).size
5354           for lg, rg in pairs(sea_ranges) do
5355             if last_char.char > rg[1] and last_char.char < rg[2] then
5356               lg = lg:sub(1, 4) ^% Remove trailing number of, eg, Cyril1
5357               local intraspace = Babel.intraspaces[lg]
5358               local intrapenalty = Babel.intrapenalties[lg]
5359               local n
5360               if intrapenalty ~= 0 then
5361                 n = node.new(14, 0)      ^% penalty
5362                 n.penalty = intrapenalty
5363                 node.insert_before(head, item, n)
5364               end
5365               n = node.new(12, 13)      ^% (glue, spaceskip)
5366               node.setglue(n, intraspace.b * quad,
5367                 intraspace.p * quad,
5368                 intraspace.m * quad)
5369               node.insert_before(head, item, n)
5370               node.remove(head, item)

```

```

5371         end
5372     end
5373 end
5374 end
5375 end
5376 }^^
5377 \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.

```

5378 \catcode`\%=14
5379 \gdef\bbl@cjkintraspacespace{%
5380 \let\bbl@cjkintraspacespace\relax
5381 \directlua{
5382     Babel = Babel or {}
5383     require('babel-data-cjk.lua')
5384     Babel.cjk_enabled = true
5385     function Babel.cjk_linebreak(head)
5386         local GLYPH = node.id'glyph'
5387         local last_char = nil
5388         local quad = 655360      % 10 pt = 655360 = 10 * 65536
5389         local last_class = nil
5390         local last_lang = nil
5391
5392         for item in node.traverse(head) do
5393             if item.id == GLYPH then
5394
5395                 local lang = item.lang
5396
5397                 local LOCALE = node.get_attribute(item,
5398                     Babel.attr_locale)
5399                 local props = Babel.locale_props[LOCALE]
5400
5401                 local class = Babel.cjk_class[item.char].c
5402
5403                 if props.cjk_quotes and props.cjk_quotes[item.char] then
5404                     class = props.cjk_quotes[item.char]
5405                 end
5406
5407                 if class == 'cp' then class = 'cl' end % ]) as CL
5408                 if class == 'id' then class = 'I' end
5409
5410                 local br = 0
5411                 if class and last_class and Babel.cjk_breaks[last_class][class] then
5412                     br = Babel.cjk_breaks[last_class][class]
5413                 end
5414
5415                 if br == 1 and props.linebreak == 'c' and
5416                     lang ~= \the\l@nohyphenation\space and
5417                     last_lang ~= \the\l@nohyphenation then
5418                     local intrapenalty = props.intrapenalty
5419                     if intrapenalty ~= 0 then
5420                         local n = node.new(14, 0)      % penalty
5421                         n.penalty = intrapenalty
5422                         node.insert_before(head, item, n)
5423                     end
5424                     local intraspacespace = props.intraspacespace

```

```

5425         local n = node.new(12, 13)      % (glue, spaceskip)
5426         node.setglue(n, intraspaces.b * quad,
5427                     intraspaces.p * quad,
5428                     intraspaces.m * quad)
5429         node.insert_before(head, item, n)
5430     end
5431
5432     if font.getfont(item.font) then
5433         quad = font.getfont(item.font).size
5434     end
5435     last_class = class
5436     last_lang = lang
5437     else % if penalty, glue or anything else
5438         last_class = nil
5439     end
5440 end
5441 lang.hyphenate(head)
5442 end
5443 }%
5444 \bbl@luahyphenate}
5445 \gdef\bbl@luahyphenate{%
5446 \let\bbl@luahyphenate\relax
5447 \directlua{
5448     luatexbase.add_to_callback('hyphenate',
5449     function (head, tail)
5450         if Babel.linebreaking.before then
5451             for k, func in ipairs(Babel.linebreaking.before) do
5452                 func(head)
5453             end
5454         end
5455         if Babel.cjk_enabled then
5456             Babel.cjk_linebreak(head)
5457         end
5458         lang.hyphenate(head)
5459         if Babel.linebreaking.after then
5460             for k, func in ipairs(Babel.linebreaking.after) do
5461                 func(head)
5462             end
5463         end
5464         if Babel.sea_enabled then
5465             Babel.sea_disc_to_space(head)
5466         end
5467     end,
5468     'Babel.hyphenate')
5469 }
5470 }
5471 \endgroup
5472 \def\bbl@provide@intraspace{%
5473 \bbl@ifunset{\bbl@intsp@\languagename}{}%
5474 {\expandafter\ifx\cscname\bbl@intsp@\languagename\endcscname\@empty\else
5475 \bbl@xin@{/c}{/\bbl@cl{\lnbrk}}}%
5476 \ifin@ % cjk
5477 \bbl@cjk_intraspace
5478 \directlua{
5479     Babel = Babel or {}
5480     Babel.locale_props = Babel.locale_props or {}
5481     Babel.locale_props[\the\localeid].linebreak = 'c'
5482 }%
5483 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@%
5484 \ifx\bbl@KVP@intrapenalty\@nnil
5485 \bbl@intrapenalty0\@%
5486 \fi
5487 \else % sea

```

```

5488 \bbl@seaintraspace
5489 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@}%
5490 \directlua{
5491   Babel = Babel or {}
5492   Babel.sea_ranges = Babel.sea_ranges or {}
5493   Babel.set_chranges('\bbl@cl{sbcpr}',
5494                     '\bbl@cl{chrng}')
5495 }%
5496 \ifx\bbl@KVP@intrapenalty\@nnil
5497   \bbl@intrapenalty0\@
5498 \fi
5499 \fi
5500 \fi
5501 \ifx\bbl@KVP@intrapenalty\@nnil\else
5502   \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
5503 \fi}}

```

9.7 Arabic justification

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

```

5504 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5505 \def\bblar@chars{%
5506   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5507   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5508   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5509 \def\bblar@elongated{%
5510   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5511   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5512   0649,064A}
5513 \begingroup
5514 \catcode`_ =11 \catcode`:=11
5515 \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5516 \endgroup
5517 \gdef\bblar@arabicjust{% TODO. Allow for several locales.
5518   \let\bblar@arabicjust\relax
5519   \newattribute\bblar@kashida
5520   \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5521   \bblar@kashida=\z@
5522   \bbl@patchfont{\bbl@parsejalt}}%
5523   \directlua{
5524     Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5525     Babel.arabic.elong_map[\the\localeid] = {}
5526     luatexbase.add_to_callback('post_linebreak_filter',
5527                               Babel.arabic.justify, 'Babel.arabic.justify')
5528     luatexbase.add_to_callback('hpack_filter',
5529                               Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5530   }}%

```

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

```

5531 \def\bblar@fetchjalt#1#2#3#4{%
5532   \bbl@exp{\bbl@foreach{#1}}{%
5533     \bbl@ifunset\bblar@JE@##1}%
5534     {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"##1#2}}%
5535     {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"\@nameuse\bblar@JE@##1#2}}}%
5536 \directlua{%
5537   local last = nil
5538   for item in node.traverse(tex.box[0].head) do
5539     if item.id == node.id'glyph' and item.char > 0x600 and
5540        not (item.char == 0x200D) then
5541       last = item
5542     end
5543   end

```

```

5544     Babel.arabic.#3['##1#4'] = last.char
5545     }}}

```

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

```

5546 \gdef\bbl@parsejalt{%
5547   \ifx\addfontfeature\@undefined\else
5548     \bbl@xin@{/e}{/\bbl@cl{lbrk}}%
5549     \ifin@
5550     \directlua{%
5551       if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5552         Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5553         tex.print([[string\cname\space bbl@parsejalti\endcname]])
5554       end
5555     }%
5556   \fi
5557 \fi}
5558 \gdef\bbl@parsejalti{%
5559   \begingroup
5560     \let\bbl@parsejalt\relax    % To avoid infinite loop
5561     \edef\bbl@tempb{\fontid\font}%
5562     \bblar@nofswarn
5563     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5564     \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
5565     \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
5566     \addfontfeature{RawFeature+=jalt}%
5567     % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5568     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5569     \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
5570     \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
5571     \directlua{%
5572       for k, v in pairs(Babel.arabic.from) do
5573         if Babel.arabic.dest[k] and
5574           not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5575           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5576             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5577         end
5578       end
5579     }%
5580   \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```

5581 \begingroup
5582 \catcode`#=11
5583 \catcode`~ =11
5584 \directlua{
5585
5586 Babel.arabic = Babel.arabic or {}
5587 Babel.arabic.from = {}
5588 Babel.arabic.dest = {}
5589 Babel.arabic.justify_factor = 0.95
5590 Babel.arabic.justify_enabled = true
5591 Babel.arabic.kashida_limit = -1
5592
5593 function Babel.arabic.justify(head)
5594   if not Babel.arabic.justify_enabled then return head end
5595   for line in node.traverse_id(node.id'hlist', head) do
5596     Babel.arabic.justify_hlist(head, line)
5597   end
5598   return head
5599 end
5600
5601 function Babel.arabic.justify_hbox(head, gc, size, pack)
5602   local has_inf = false

```

```

5603 if Babel.arabic.justify_enabled and pack == 'exactly' then
5604     for n in node.traverse_id(12, head) do
5605         if n.stretch_order > 0 then has_inf = true end
5606     end
5607     if not has_inf then
5608         Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5609     end
5610 end
5611 return head
5612 end
5613
5614 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5615     local d, new
5616     local k_list, k_item, pos_inline
5617     local width, width_new, full, k_curr, wt_pos, goal, shift
5618     local subst_done = false
5619     local elong_map = Babel.arabic.elong_map
5620     local cnt
5621     local last_line
5622     local GLYPH = node.id'glyph'
5623     local KASHIDA = Babel.attr_kashida
5624     local LOCALE = Babel.attr_locale
5625
5626     if line == nil then
5627         line = {}
5628         line.glue_sign = 1
5629         line.glue_order = 0
5630         line.head = head
5631         line.shift = 0
5632         line.width = size
5633     end
5634
5635     % Exclude last line. todo. But-- it discards one-word lines, too!
5636     % ? Look for glue = 12:15
5637     if (line.glue_sign == 1 and line.glue_order == 0) then
5638         elongs = {}      % Stores elongated candidates of each line
5639         k_list = {}      % And all letters with kashida
5640         pos_inline = 0  % Not yet used
5641
5642         for n in node.traverse_id(GLYPH, line.head) do
5643             pos_inline = pos_inline + 1 % To find where it is. Not used.
5644
5645             % Elongated glyphs
5646             if elong_map then
5647                 local locale = node.get_attribute(n, LOCALE)
5648                 if elong_map[locale] and elong_map[locale][n.font] and
5649                     elong_map[locale][n.font][n.char] then
5650                     table.insert(elongs, {node = n, locale = locale} )
5651                     node.set_attribute(n.prev, KASHIDA, 0)
5652                 end
5653             end
5654
5655             % Tatwil
5656             if Babel.kashida_wts then
5657                 local k_wt = node.get_attribute(n, KASHIDA)
5658                 if k_wt > 0 then % todo. parameter for multi inserts
5659                     table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5660                 end
5661             end
5662
5663         end % of node.traverse_id
5664
5665         if #elongs == 0 and #k_list == 0 then goto next_line end

```

```

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

```

```

5729         cnt = cnt + 1
5730         if cnt > Babel.arabic.kashida_limit then
5731             node.remove(line.head, n)
5732         end
5733         else
5734             cnt = 0
5735         end
5736     end
5737 end
5738
5739 ::next_line::
5740
5741 % Must take into account marks and ins, see luatex manual.
5742 % Have to be executed only if there are changes. Investigate
5743 % what's going on exactly.
5744 if subst_done and not gc then
5745     d = node.hpack(line.head, full, 'exactly')
5746     d.shift = shift
5747     node.insert_before(head, line, d)
5748     node.remove(head, line)
5749 end
5750 end % if process line
5751 end
5752 }
5753 \endgroup
5754 \fi\fi % ends Arabic just block: \ifnum\bbI@bidimode>100...

```

9.8 Common stuff

```

5755 \AddBabelHook{babel-fontspec}{afterextras}{\bbI@switchfont}
5756 \AddBabelHook{babel-fontspec}{beforestart}{\bbI@cckstdfont}
5757 \DisableBabelHook{babel-fontspec}
5758 <<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.

```

5759 % TODO - to a lua file
5760 \directlua{
5761 Babel.script_blocks = {
5762   ['dflt'] = {},
5763   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5764             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5765   ['Armn'] = {{0x0530, 0x058F}},
5766   ['Beng'] = {{0x0980, 0x09FF}},
5767   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5768   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5769   ['Cyril'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5770             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5771   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5772   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5773             {0xAB00, 0xAB2F}},
5774   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5775   % Don't follow strictly Unicode, which places some Coptic letters in
5776   % the 'Greek and Coptic' block
5777   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5778   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF}},

```



```

5779         {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5780         {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5781         {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5782         {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5783         {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5784 ['Hebr'] = {{0x0590, 0x05FF}},
5785 ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5786         {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5787 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5788 ['Knda'] = {{0x0C80, 0x0CFF}},
5789 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5790         {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5791         {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5792 ['Lao'] = {{0x0E80, 0x0EFF}},
5793 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5794         {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5795         {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5796 ['Mahj'] = {{0x1150, 0x117F}},
5797 ['Mlym'] = {{0x0D00, 0x0D7F}},
5798 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5799 ['Orya'] = {{0x0B00, 0x0B7F}},
5800 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x11E0, 0x11FF}},
5801 ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5802 ['Taml'] = {{0x0B80, 0x0BFF}},
5803 ['Telu'] = {{0x0C00, 0x0C7F}},
5804 ['Tfng'] = {{0x2D30, 0x2D7F}},
5805 ['Thai'] = {{0x0E00, 0x0E7F}},
5806 ['Tibt'] = {{0x0F00, 0x0FFF}},
5807 ['Vaii'] = {{0xA500, 0xA63F}},
5808 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5809 }
5810
5811 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5812 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5813 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5814
5815 function Babel.locale_map(head)
5816   if not Babel.locale_mapped then return head end
5817
5818   local LOCALE = Babel.attr_locale
5819   local GLYPH = node.id('glyph')
5820   local inmath = false
5821   local toloc_save
5822   for item in node.traverse(head) do
5823     local toloc
5824     if not inmath and item.id == GLYPH then
5825       % Optimization: build a table with the chars found
5826       if Babel.chr_to_loc[item.char] then
5827         toloc = Babel.chr_to_loc[item.char]
5828       else
5829         for lc, maps in pairs(Babel.loc_to_scr) do
5830           for _, rg in pairs(maps) do
5831             if item.char >= rg[1] and item.char <= rg[2] then
5832               Babel.chr_to_loc[item.char] = lc
5833               toloc = lc
5834               break
5835             end
5836           end
5837         end
5838       end
5839       % Now, take action, but treat composite chars in a different
5840       % fashion, because they 'inherit' the previous locale. Not yet
5841       % optimized.

```

```

5842     if not toloc and
5843         (item.char >= 0x0300 and item.char <= 0x036F) or
5844         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5845         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5846         toloc = toloc_save
5847     end
5848     if toloc and Babel.locale_props[toloc] and
5849         Babel.locale_props[toloc].letters and
5850         tex.getcatcode(item.char) \string~= 11 then
5851         toloc = nil
5852     end
5853     if toloc and toloc > -1 then
5854         if Babel.locale_props[toloc].lg then
5855             item.lang = Babel.locale_props[toloc].lg
5856             node.set_attribute(item, LOCALE, toloc)
5857         end
5858         if Babel.locale_props[toloc]['/'..item.font] then
5859             item.font = Babel.locale_props[toloc]['/'..item.font]
5860         end
5861         toloc_save = toloc
5862     end
5863     elseif not inmath and item.id == 7 then % Apply recursively
5864         item.replace = item.replace and Babel.locale_map(item.replace)
5865         item.pre      = item.pre and Babel.locale_map(item.pre)
5866         item.post     = item.post and Babel.locale_map(item.post)
5867     elseif item.id == node.id'math' then
5868         inmath = (item.subtype == 0)
5869     end
5870 end
5871 return head
5872 end
5873 }

```

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

```

5874 \newcommand\babelcharproperty[1]{%
5875   \count@=#1\relax
5876   \ifvmode
5877     \expandafter\bbl@chprop
5878   \else
5879     \bbl@error{\string\babelcharproperty\space can be used only in\%
5880               vertical mode (preamble or between paragraphs)}%
5881     {See the manual for futher info}%
5882   \fi}
5883 \newcommand\bbl@chprop[3][\the\count@]{%
5884   \@tempcnta=#1\relax
5885   \bbl@ifunset{bbl@chprop@#2}%
5886   {\bbl@error{No property named '#2'. Allowed values are\%
5887             direction (bc), mirror (bmg), and linebreak (lb)}%
5888     {See the manual for futher info}}%
5889   }%
5890   \loop
5891     \bbl@cs{chprop@#2}{#3}%
5892     \ifnum\count@<\@tempcnta
5893       \advance\count@\@ne
5894     \repeat}
5895 \def\bbl@chprop@direction#1{%
5896   \directlua{
5897     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5898     Babel.characters[\the\count@]['d'] = '#1'
5899   }}
5900 \let\bbl@chprop@bc\bbl@chprop@direction
5901 \def\bbl@chprop@mirror#1{%

```

```

5902 \directlua{
5903   Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5904   Babel.characters[\the\count@]['m'] = '\number#1'
5905 }
5906 \let\bbl@chprop@bmg\bbl@chprop@mirror
5907 \def\bbl@chprop@linebreak#1{%
5908   \directlua{
5909     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5910     Babel.cjk_characters[\the\count@]['c'] = '#1'
5911   }}
5912 \let\bbl@chprop@lb\bbl@chprop@linebreak
5913 \def\bbl@chprop@locale#1{%
5914   \directlua{
5915     Babel.chr_to_loc = Babel.chr_to_loc or {}
5916     Babel.chr_to_loc[\the\count@] =
5917       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
5918   }}

```

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.

```

5919 \directlua{
5920   Babel.nohyphenation = \the\l@nohyphenation
5921 }

```

Now the TeX high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the $\{n\}$ syntax. For example, $\text{pre}=\{1\}\{1\}$ becomes $\text{function}(m) \text{return } m[1]..m[1]..'-' \text{end}$, where m are the matches returned after applying the pattern. With a mapped capture the functions are similar to $\text{function}(m) \text{return } \text{Babel.capt_map}(m[1], 1) \text{end}$, where the last argument identifies the mapping to be applied to $m[1]$. The way it is carried out is somewhat tricky, but the effect is not dissimilar to lua load – save the code as string in a TeX macro, and expand this macro at the appropriate place. As \directlua does not take into account the current catcode of $@$, we just avoid this character in macro names (which explains the internal group, too).

```

5922 \begingroup
5923 \catcode`\~ = 12
5924 \catcode`\% = 12
5925 \catcode`\& = 14
5926 \catcode`\| = 12
5927 \gdef\babelprehyphenation{%&
5928   \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]]}
5929 \gdef\babelposthyphenation{%&
5930   \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]]}
5931 \gdef\bbl@settransform#1[#2]#3#4#5{%&
5932   \ifcase#1
5933     \bbl@activateprehyphen
5934   \or
5935     \bbl@activateposthyphen
5936   \fi
5937 \begingroup
5938   \def\babeltempa{\bbl@add@list\babeltempb}&%
5939   \let\babeltempb@empty
5940   \def\bbl@tempa{#5}&%
5941   \bbl@replace\bbl@tempa{,}{ ,}&% TODO. Ugly trick to preserve {}
5942   \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5943     \bbl@ifsamestring{##1}{remove}&%
5944     {\bbl@add@list\babeltempb{nil}}}&%
5945   {\directlua{
5946     local rep = [=[#1]=]
5947     rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5948     rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5949     rep = rep:gsub('(string)%s*=%s*([%s,]*)', Babel.capture_func)
5950     if #1 == 0 or #1 == 2 then
5951       rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5952         'space = {' .. '%2, %3, %4' .. '}')

```

```

5953         rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5954         'spacefactor = { ' .. '%2, %3, %4' .. ' }')
5955         rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
5956     else
5957         rep = rep:gsub(      '(no)%s*=%s*([^\s,]*)', Babel.capture_func)
5958         rep = rep:gsub(      '(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5959         rep = rep:gsub(      '(post)%s*=%s*([^\s,]*)', Babel.capture_func)
5960     end
5961     tex.print([[string\babeltempa{[] .. rep .. [{}]])
5962     ]]}&%
5963 \bbl@foreach\babeltempb{&%
5964   \bbl@forkv{##1}}{&%
5965     \in{,###1,}{,nil,step,data,remove,insert,string,no,pre,&%
5966       no,post,penalty,kashida,space,spacefactor,}&%
5967     \ifin@else
5968       \bbl@error
5969         {Bad option '###1' in a transform.\\&%
5970          I'll ignore it but expect more errors}&%
5971         {See the manual for further info.}&%
5972     \fi}}&%
5973 \let\bbl@kv@attribute\relax
5974 \let\bbl@kv@label\relax
5975 \let\bbl@kv@fonts\@empty
5976 \bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}&%
5977 \ifx\bbl@kv@fonts\@empty\else\bbl@settransfont\fi
5978 \ifx\bbl@kv@attribute\relax
5979   \ifx\bbl@kv@label\relax\else
5980     \bbl@exp{\bbl@trim@def\bbl@kv@fonts{\bbl@kv@fonts}}&%
5981     \bbl@replace\bbl@kv@fonts{ }{,}&%
5982     \edef\bbl@kv@attribute{\bbl@ATR@\bbl@kv@label @#3@\bbl@kv@fonts}&%
5983     \count@\z@
5984     \def\bbl@elt##1##2##3{&%
5985       \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
5986       {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
5987         {\count@\@ne}&%
5988         {\bbl@error
5989           {Transforms cannot be re-assigned to different\\&%
5990            fonts. The conflict is in '\bbl@kv@label'.\\&%
5991            Apply the same fonts or use a different label}&%
5992            {See the manual for further details.}}}&%
5993         }}&%
5994     \bbl@transfont@list
5995     \ifnum\count@=\z@
5996       \bbl@exp{\global\bbl@add\bbl@transfont@list
5997         {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
5998     \fi
5999     \bbl@ifunset{\bbl@kv@attribute}&%
6000     {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6001     {}&%
6002     \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
6003   \fi
6004 \else
6005   \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6006 \fi
6007 \directlua{
6008   local lbkr = Babel.linebreaking.replacements[#1]
6009   local u = unicode.utf8
6010   local id, attr, label
6011   if #1 == 0 then
6012     id = \the\csname bbl@id@#3\endcsname\space
6013   else
6014     id = \the\csname l@#3\endcsname\space
6015   end

```

```

6016 \ifx\bbl@kv@attribute\relax
6017   attr = -1
6018 \else
6019   attr = luatexbase.registernumber'\bbl@kv@attribute'
6020 \fi
6021 \ifx\bbl@kv@label\relax\else &% Same refs:
6022   label = [==[\bbl@kv@label]==]
6023 \fi
6024 &% Convert pattern:
6025 local patt = string.gsub([=#4]=], '%s', '')
6026 if #1 == 0 then
6027   patt = string.gsub(patt, '|', ' ')
6028 end
6029 if not u.find(patt, '()', nil, true) then
6030   patt = '()' .. patt .. '()'
6031 end
6032 if #1 == 1 then
6033   patt = string.gsub(patt, '%(%)^', '^()')
6034   patt = string.gsub(patt, '%$%(%)', '()$')
6035 end
6036 patt = u.gsub(patt, '{(.)}',
6037   function (n)
6038     return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6039   end)
6040 patt = u.gsub(patt, '{(%x%x%x%x+)}',
6041   function (n)
6042     return u.gsub(u.char(tonumber(n, 16)), '(%)', '%%1')
6043   end)
6044 lbkr[id] = lbkr[id] or {}
6045 table.insert(lbkr[id],
6046   { label=label, attr=attr, pattern=patt, replace={\labeltempb} })
6047 }&%
6048 \endgroup}
6049 \endgroup
6050 \let\bbl@transfont@list@empty
6051 \def\bbl@settransfont{%
6052   \global\let\bbl@settransfont\relax % Execute only once
6053   \gdef\bbl@transfont{%
6054     \def\bbl@elt####1####2####3{%
6055       \bbl@ifblank{####3}%
6056         {\count@tw@}% Do nothing if no fonts
6057         {\count@z@
6058           \bbl@vforeach{####3}{%
6059             \def\bbl@tempd{#####1}%
6060             \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6061             \ifx\bbl@tempd\bbl@tempe
6062               \count@one
6063             \else\ifx\bbl@tempd\bbl@transfam
6064               \count@one
6065             \fi\fi}%
6066           \ifcase\count@
6067             \bbl@csarg\unsetattribute{ATR@####2@####1@####3}%
6068           \or
6069             \bbl@csarg\setattribute{ATR@####2@####1@####3}\@one
6070           \fi}}%
6071     \bbl@transfont@list}%
6072 \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6073 \gdef\bbl@transfam{-unknown-}%
6074 \bbl@foreach\bbl@font@fams{%
6075   \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6076   \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6077   {\xdef\bbl@transfam{##1}}%
6078   {}}

```

```

6079 \DeclareRobustCommand\enablelocaletransform[1]{%
6080   \bbl@ifunset{bbl@ATR@#1@\languagename @}%
6081   {\bbl@error
6082     {'#1' for '\languagename' cannot be enabled.\\%
6083     Maybe there is a typo or it's a font-dependent transform}%
6084     {See the manual for further details.}}%
6085   {\bbl@csarg\setattribute{ATR@#1@\languagename @}\@ne}}
6086 \DeclareRobustCommand\disablelocaletransform[1]{%
6087   \bbl@ifunset{bbl@ATR@#1@\languagename @}%
6088   {\bbl@error
6089     {'#1' for '\languagename' cannot be disabled.\\%
6090     Maybe there is a typo or it's a font-dependent transform}%
6091     {See the manual for further details.}}%
6092   {\bbl@csarg\unsetattribute{ATR@#1@\languagename @}}}
6093 \def\bbl@activateposthyphen{%
6094   \let\bbl@activateposthyphen\relax
6095   \directlua{
6096     require('babel-transforms.lua')
6097     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6098   }}
6099 \def\bbl@activateprehyphen{%
6100   \let\bbl@activateprehyphen\relax
6101   \directlua{
6102     require('babel-transforms.lua')
6103     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6104   }}

```

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

```

6105 \newcommand\localeprehyphenation[1]{%
6106   \directlua{ Babel.string_prehyphenation(=[#1]=, \the\localeid) }}

```

9.10 Bidi

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

```

6107 \def\bbl@activate@preotf{%
6108   \let\bbl@activate@preotf\relax % only once
6109   \directlua{
6110     Babel = Babel or {}
6111     %
6112     function Babel.pre_otfload_v(head)
6113       if Babel.numbers and Babel.digits_mapped then
6114         head = Babel.numbers(head)
6115       end
6116       if Babel.bidi_enabled then
6117         head = Babel.bidi(head, false, dir)
6118       end
6119       return head
6120     end
6121     %
6122     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6123       if Babel.numbers and Babel.digits_mapped then
6124         head = Babel.numbers(head)
6125       end
6126       if Babel.bidi_enabled then
6127         head = Babel.bidi(head, false, dir)
6128       end
6129       return head

```

```

6130 end
6131 %
6132 luatexbase.add_to_callback('pre_linebreak_filter',
6133   Babel.pre_otfload_v,
6134   'Babel.pre_otfload_v',
6135   luatexbase.priority_in_callback('pre_linebreak_filter',
6136     'luaotfload.node_processor') or nil)
6137 %
6138 luatexbase.add_to_callback('hpack_filter',
6139   Babel.pre_otfload_h,
6140   'Babel.pre_otfload_h',
6141   luatexbase.priority_in_callback('hpack_filter',
6142     'luaotfload.node_processor') or nil)
6143 }}

```

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=`.

```

6144 \breakafterdirmode=1
6145 \ifnum\bbl@bidimode>\@ne % Any bidi= except default=1
6146   \let\bbl@beforeforeign\leavevmode
6147   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6148   \RequirePackage{luatexbase}
6149   \bbl@activate@preotf
6150   \directlua{
6151     require('babel-data-bidi.lua')
6152     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6153       require('babel-bidi-basic.lua')
6154     \or
6155       require('babel-bidi-basic-r.lua')
6156     \fi}
6157   \newattribute\bbl@attr@dir
6158   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6159   \bbl@exp{\output{\bodydir\pagedir\the\output}}
6160 \fi
6161 \chardef\bbl@thetextdir\z@
6162 \chardef\bbl@thepardir\z@
6163 \def\bbl@getluadir#1{%
6164   \directlua{
6165     if tex.#1dir == 'TLT' then
6166       tex.sprint('0')
6167     elseif tex.#1dir == 'TRT' then
6168       tex.sprint('1')
6169     end}}
6170 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6171   \ifcase#3\relax
6172     \ifcase\bbl@getluadir{#1}\relax\else
6173       #2 TLT\relax
6174     \fi
6175   \else
6176     \ifcase\bbl@getluadir{#1}\relax
6177       #2 TRT\relax
6178     \fi
6179   \fi}
6180% ..00PPTT, with masks 0xC (par dir) and 0x3 (text dir)
6181 \def\bbl@thedir{0}
6182 \def\bbl@textdir#1{%
6183   \bbl@setluadir{text}\textdir{#1}%
6184   \chardef\bbl@thetextdir#1\relax
6185   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6186   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6187 \def\bbl@pardir#1{% Used twice
6188   \bbl@setluadir{par}\pardir{#1}%

```

```

6189 \chardef\bbl@thepardir#1\relax}
6190 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}% Used once
6191 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}% Unused
6192 \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.

```

6193 \ifnum\bbl@bidimode>\z@ % Any bidi=
6194 \def\bbl@insidemath{0}%
6195 \def\bbl@everymath{\def\bbl@insidemath{1}}
6196 \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6197 \frozen@everymath\expandafter{%
6198   \expandafter\bbl@everymath\the\frozen@everymath}
6199 \frozen@everydisplay\expandafter{%
6200   \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6201 \AtBeginDocument{
6202   \directlua{
6203     function Babel.math_box_dir(head)
6204       if not (token.get_macro('bbl@insidemath') == '0') then
6205         if Babel.hlist_has_bidi(head) then
6206           local d = node.new(node.id'dir')
6207           d.dir = '+TRT'
6208           node.insert_before(head, node.has_glyph(head), d)
6209           for item in node.traverse(head) do
6210             node.set_attribute(item,
6211               Babel.attr_dir, token.get_macro('bbl@thedir'))
6212           end
6213         end
6214       end
6215       return head
6216     end
6217     luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6218       "Babel.math_box_dir", 0)
6219   }}%
6220 \fi

```

9.11 Layout

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

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

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

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

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

```

6221 \bbl@trace{Redefinitions for bidi layout}
6222 %
6223 <<{*More package options}>> ≡
6224 \chardef\bbl@eqnpos\z@
6225 \DeclareOption{leqno}{\chardef\bbl@eqnpos@ne}
6226 \DeclareOption{fleqn}{\chardef\bbl@eqnpos@tw@}

```



```

6227 <</More package options>>
6228 %
6229 \ifnum\bb@bidimode>\z@ % Any bidi=
6230 \matheqdirmode\@ne % A luatex primitive
6231 \let\bb@eqnodir\relax
6232 \def\bb@eqdel{()}
6233 \def\bb@eqnum{%
6234   {\normalfont\normalcolor
6235     \expandafter\@firstoftwo\bb@eqdel
6236     \theequation
6237     \expandafter\@secondoftwo\bb@eqdel}}
6238 \def\bb@puteqno#1{\eqno\hbox{#1}}
6239 \def\bb@putleqno#1{\leqno\hbox{#1}}
6240 \def\bb@eqno@flip#1{%
6241   \ifdim\predisplaysize=-\maxdimen
6242     \eqno
6243     \hb@xt@.01pt{%
6244       \hb@xt@\displaywidth{\hss{#1}\glet\bb@upset\@currentlabel}\hss}%
6245   \else
6246     \leqno\hbox{#1}\glet\bb@upset\@currentlabel}%
6247   \fi
6248   \bb@exp{\def\@currentlabel{\[bb@upset]}}
6249 \def\bb@leqno@flip#1{%
6250   \ifdim\predisplaysize=-\maxdimen
6251     \leqno
6252     \hb@xt@.01pt{%
6253       \hss\hb@xt@\displaywidth{\#1\glet\bb@upset\@currentlabel}\hss}%
6254   \else
6255     \eqno\hbox{#1}\glet\bb@upset\@currentlabel}%
6256   \fi
6257   \bb@exp{\def\@currentlabel{\[bb@upset]}}
6258 \AtBeginDocument{%
6259   \ifx\bb@noamsmath\relax\else
6260   \ifx\maketag@@@\@undefined % Normal equation, eqnarray
6261     \AddToHook{env/equation/begin}{%
6262       \ifnum\bb@thetextdir>\z@
6263         \def\bb@mathboxdir{\def\bb@insidemath{1}}%
6264         \let\@eqnum\bb@eqnum
6265         \edef\bb@eqnodir{\noexpand\bb@textdir{\the\bb@thetextdir}}%
6266         \chardef\bb@thetextdir\z@
6267         \bb@add\normalfont{\bb@eqnodir}%
6268         \ifcase\bb@eqnpos
6269           \let\bb@puteqno\bb@eqno@flip
6270         \or
6271           \let\bb@puteqno\bb@leqno@flip
6272         \fi
6273       \fi}%
6274   \ifnum\bb@eqnpos=\tw@\else
6275     \def\endequation{\bb@puteqno{\@eqnum}\$@\@ignoretrue}%
6276   \fi
6277   \AddToHook{env/eqnarray/begin}{%
6278     \ifnum\bb@thetextdir>\z@
6279       \def\bb@mathboxdir{\def\bb@insidemath{1}}%
6280       \edef\bb@eqnodir{\noexpand\bb@textdir{\the\bb@thetextdir}}%
6281       \chardef\bb@thetextdir\z@
6282       \bb@add\normalfont{\bb@eqnodir}%
6283     \ifnum\bb@eqnpos=\@ne
6284       \def\@eqnum{%
6285         \setbox\z@\hbox{\bb@eqnum}%
6286         \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6287     \else
6288       \let\@eqnum\bb@eqnum
6289     \fi

```

```

6290     \fi}
6291     % Hack. YA luatex bug?:
6292     \expandafter\bbledsreplace\csname] \endcsname{${$}\eqno\kern.001pt${$}}%
6293 \else % amstex
6294     \bbledexp{% Hack to hide maybe undefined conditionals:
6295         \chardef\bbledeqnpos=0%
6296         \<iftagsleft@>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6297     \ifnum\bbledeqnpos=\@ne
6298         \let\bbledams@lap\hbox
6299     \else
6300         \let\bbledams@lap\llap
6301     \fi
6302     \ExplSyntaxOn % Required by \bbledsreplace with \intertext@
6303     \bbledsreplace\intertext@{\normalbaselines}%
6304         {\normalbaselines
6305         \ifx\bbledeqnodir\relax\else\bbled@pardir\@ne\bbled@eqnodir\fi}%
6306     \ExplSyntaxOff
6307     \def\bbledams@tagbox#1#2{#1{\bbled@eqnodir#2}}% #1=hbox|lap|flip
6308     \ifx\bbledams@lap\hbox % leqno
6309         \def\bbledams@flip#1{%
6310             \hbox to 0.01pt{\hss\hbox to\displaywidth{{#1}\hss}}}%
6311     \else % eqno
6312         \def\bbledams@flip#1{%
6313             \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}\hss}}%
6314         \fi
6315     \def\bbledams@preset#1{%
6316         \def\bbled@mathboxdir{\def\bbled@insidemath{1}}%
6317         \ifnum\bbled@thetextdir>\z@
6318             \edef\bbled@eqnodir{\noexpand\bbled@textdir{\the\bbled@thetextdir}}%
6319             \bbled@sreplace\textdef@{\hbox}{\bbledams@tagbox\hbox}%
6320             \bbled@sreplace\maketag@@@{\hbox}{\bbledams@tagbox#1}%
6321         \fi}%
6322     \ifnum\bbledeqnpos=\tw@ \else
6323         \def\bbledams@equation{%
6324             \def\bbled@mathboxdir{\def\bbled@insidemath{1}}%
6325             \ifnum\bbled@thetextdir>\z@
6326                 \edef\bbled@eqnodir{\noexpand\bbled@textdir{\the\bbled@thetextdir}}%
6327                 \chardef\bbled@thetextdir\z@
6328                 \bbled@add\normalfont{\bbled@eqnodir}%
6329                 \ifcase\bbled@eqnpos
6330                     \def\veqno##1##2{\bbled@eqno@flip{##1##2}}%
6331                 \or
6332                     \def\veqno##1##2{\bbled@leqno@flip{##1##2}}%
6333                 \fi
6334             \fi}%
6335         \AddToHook{env/equation/begin}{\bbledams@equation}%
6336         \AddToHook{env/equation*/begin}{\bbledams@equation}%
6337     \fi
6338     \AddToHook{env/cases/begin}{\bbledams@preset\bbledams@lap}%
6339     \AddToHook{env/multline/begin}{\bbledams@preset\hbox}%
6340     \AddToHook{env/gather/begin}{\bbledams@preset\bbledams@lap}%
6341     \AddToHook{env/gather*/begin}{\bbledams@preset\bbledams@lap}%
6342     \AddToHook{env/align/begin}{\bbledams@preset\bbledams@lap}%
6343     \AddToHook{env/align*/begin}{\bbledams@preset\bbledams@lap}%
6344     \AddToHook{env/alignat/begin}{\bbledams@preset\bbledams@lap}%
6345     \AddToHook{env/alignat*/begin}{\bbledams@preset\bbledams@lap}%
6346     \AddToHook{env/eqnalign/begin}{\bbledams@preset\hbox}%
6347     % Hackish, for proper alignment. Don't ask me why it works!:%
6348     \bbledexp{% Avoid a 'visible' conditional
6349         \\ \AddToHook{env/align*/end}{\<iftag@>\<else>\\ \tag*{} \<fi>}%
6350         \\ \AddToHook{env/alignat*/end}{\<iftag@>\<else>\\ \tag*{} \<fi>}%
6351     \AddToHook{env/flalign/begin}{\bbledams@preset\hbox}%
6352     \AddToHook{env/split/before}{%

```

```

6353     \def\bb@mathboxdir{\def\bb@insidemath{1}}%
6354     \ifnum\bb@thetextdir>\z@
6355     \bb@ifsamestring\@currentvir{equation}%
6356     {\ifx\bb@ams@lap\hbox % leqno
6357     \def\bb@ams@flip#1{%
6358     \hbox to 0.01pt{\hbox to\displaywidth{#{1}\hss}\hss}}%
6359     \else
6360     \def\bb@ams@flip#1{%
6361     \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#{1}}}}%
6362     \fi}%
6363     }%
6364     \fi}%
6365     \fi\fi}
6366 \fi
6367 \def\bb@provide@extra#1{%
6368 % == Counters: mapdigits ==
6369 % Native digits
6370 \ifx\bb@KVP@mapdigits\@nnil\else
6371 \bb@ifunset{\bb@dgnat@\languagename}{}%
6372 {\RequirePackage{luatexbase}%
6373 \bb@activate@preotf
6374 \directlua{
6375     Babel = Babel or {} %%% -> presets in luababel
6376     Babel.digits_mapped = true
6377     Babel.digits = Babel.digits or {}
6378     Babel.digits[\the\localeid] =
6379     table.pack(string.utfvalue('\bb@cl{dgnat}'))
6380     if not Babel.numbers then
6381     function Babel.numbers(head)
6382     local LOCALE = Babel.attr_locale
6383     local GLYPH = node.id'glyph'
6384     local inmath = false
6385     for item in node.traverse(head) do
6386     if not inmath and item.id == GLYPH then
6387     local temp = node.get_attribute(item, LOCALE)
6388     if Babel.digits[temp] then
6389     local chr = item.char
6390     if chr > 47 and chr < 58 then
6391     item.char = Babel.digits[temp][chr-47]
6392     end
6393     end
6394     elseif item.id == node.id'math' then
6395     inmath = (item.subtype == 0)
6396     end
6397     end
6398     return head
6399     end
6400     end
6401     }}%
6402 \fi
6403 % == transforms ==
6404 \ifx\bb@KVP@transforms\@nnil\else
6405 \def\bb@elt##1##2##3{%
6406 \in@{transforms.}{##1}%
6407 \ifin@
6408 \def\bb@tempa{##1}%
6409 \bb@replace\bb@tempa{transforms.}{}%
6410 \bb@carg\bb@transforms{babel\bb@tempa}{##2}{##3}%
6411 \fi}%
6412 \csname \bb@inidata@\languagename\endcsname
6413 \bb@release@transforms\relax % \relax closes the last item.
6414 \fi}
6415 % Start tabular here:

```

```

6416 \def\localerestoredirs{%
6417   \ifcase\bbbl@thetextdir
6418     \ifnum\textdirection=\z@\else\textdir TLT\fi
6419   \else
6420     \ifnum\textdirection=\@ne\else\textdir TRT\fi
6421   \fi
6422   \ifcase\bbbl@thepardir
6423     \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6424   \else
6425     \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6426   \fi}
6427 \IfBabelLayout{tabular}%
6428   {\chardef\bbbl@tabular@mode\tw@}% All RTL
6429   {\IfBabelLayout{notabular}%
6430     {\chardef\bbbl@tabular@mode\z@}%
6431     {\chardef\bbbl@tabular@mode\@ne}}% Mixed, with LTR cols
6432 \ifnum\bbbl@bidimode>\@ne % Any lua bidi= except default=1
6433 \ifcase\bbbl@tabular@mode\or % 1
6434   \let\bbbl@parabefore\relax
6435   \AddToHook{para/before}{\bbbl@parabefore}
6436   \AtBeginDocument{%
6437     \bbbl@replace\@tabular{${}$}%
6438     \def\bbbl@insidemath{0}%
6439     \def\bbbl@parabefore{\localerestoredirs}}%
6440   \ifnum\bbbl@tabular@mode=\@ne
6441     \bbbl@ifunset{\@tabclassz}{}%
6442     \bbbl@exp{% Hide conditionals
6443       \\bbbl@sreplace\\ \@tabclassz
6444         {\<ifcase>\\ \@chnum}%
6445         {\localerestoredirs\<ifcase>\\ \@chnum}}}%
6446     \@ifpackageloaded{colortbl}%
6447       {\bbbl@sreplace\@classz
6448         {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6449       {\@ifpackageloaded{array}%
6450         {\bbbl@exp{% Hide conditionals
6451           \\bbbl@sreplace\\ \@classz
6452             {\<ifcase>\\ \@chnum}%
6453             {\bgroup\\localerestoredirs\<ifcase>\\ \@chnum}%
6454             \\bbbl@sreplace\\ \@classz
6455             {\do@row@strut\<fi>}{\do@row@strut\<fi>\egroup}}}%
6456         {}}%
6457     \fi}%
6458 \or % 2
6459   \let\bbbl@parabefore\relax
6460   \AddToHook{para/before}{\bbbl@parabefore}%
6461   \AtBeginDocument{%
6462     \@ifpackageloaded{colortbl}%
6463       {\bbbl@replace\@tabular{${}$}%
6464         \def\bbbl@insidemath{0}%
6465         \def\bbbl@parabefore{\localerestoredirs}}%
6466       \bbbl@sreplace\@classz
6467       {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6468     {}}%
6469 \fi

```

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

```

6470 \AtBeginDocument{%
6471   \@ifpackageloaded{multicol}%
6472     {\toks@\expandafter{\multi@column@out}%
6473     \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
6474   {}}%

```

```

6475 \ifpackageloaded{paracol}%
6476 {\edef\pcol@output{%
6477 \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
6478 {}}%
6479 \fi
6480 \ifx\bblopt@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. \bblnextfake 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.

6481 \ifnum\bblobidimode>\z@ % Any bidi=
6482 \def\bblnextfake#1{% non-local changes, use always inside a group!
6483 \bblexp{%
6484 \def\bblobinsidemath{0}%
6485 \mathdir\the\bodydir
6486 #1% Once entered in math, set boxes to restore values
6487 \<ifmmode>%
6488 \everyvbox{%
6489 \the\everyvbox
6490 \bodydir\the\bodydir
6491 \mathdir\the\mathdir
6492 \everyhbox{\the\everyhbox}%
6493 \everyvbox{\the\everyvbox}}%
6494 \everyhbox{%
6495 \the\everyhbox
6496 \bodydir\the\bodydir
6497 \mathdir\the\mathdir
6498 \everyhbox{\the\everyhbox}%
6499 \everyvbox{\the\everyvbox}}%
6500 \<fi>}}%
6501 \def\@hangfrom#1{%
6502 \setbox\@tempboxa\hbox{#1}%
6503 \hangindent\wd\@tempboxa
6504 \ifnum\bblobgetluadir{page}=\bblobgetluadir{par}\else
6505 \shapemode\@ne
6506 \fi
6507 \noindent\box\@tempboxa}
6508 \fi
6509 \IfBabelLayout{tabular}
6510 {\let\bblobL@tabular\@tabular
6511 \bblobreplace\@tabular{\$}\bblnextfake$}%
6512 \let\bblobNL@tabular\@tabular
6513 \AtBeginDocument{%
6514 \ifx\bblobNL@tabular\@tabular\else
6515 \bblobexp{\in{\bblnextfake}{\@tabular}}%
6516 \ifin\else
6517 \bblobreplace\@tabular{\$}\bblnextfake$}%
6518 \fi
6519 \let\bblobNL@tabular\@tabular
6520 \fi}}
6521 {}
6522 \IfBabelLayout{lists}
6523 {\let\bblobL@list\list
6524 \bblobsreplace\list{\parshape}\bbloblistparshape}%
6525 \let\bblobNL@list\list
6526 \def\bbloblistparshape#1#2#3{%
6527 \parshape #1 #2 #3 %
6528 \ifnum\bblobgetluadir{page}=\bblobgetluadir{par}\else
6529 \shapemode\tw@
6530 \fi}}
6531 {}
6532 \IfBabelLayout{graphics}

```

```

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

```

```

6596     \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\tw}%
6597     \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6598     \bbl@sreplace\tikz{\begin@group}{\begin@group\bbl@pictsetdir\tw}%
6599     \fi
6600     \ifx\tcolorbox\@undefined\else
6601     \def\tcb@drawing@env@begin{%
6602     \csname tcb@before@\tcb@split@state\endcsname
6603     \bbl@pictsetdir\tw@
6604     \begin{\kvtcb@graphenv}%
6605     \tcb@bbdraw%
6606     \tcb@apply@graph@patches
6607     }%
6608     \def\tcb@drawing@env@end{%
6609     \end{\kvtcb@graphenv}%
6610     \bbl@pictresetdir
6611     \csname tcb@after@\tcb@split@state\endcsname
6612     }%
6613     \fi
6614   }}
6615 {}

```

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`.

```

6616 \IfBabelLayout{counters*}%
6617   {\bbl@add\bbl@opt@layout{.counters.}%
6618   \directlua{
6619     luatexbase.add_to_callback("process_output_buffer",
6620     Babel.discard_sublr , "Babel.discard_sublr") }%
6621   }}
6622 \IfBabelLayout{counters}%
6623   {\let\bbl@0L@@textsuperscript\@textsuperscript
6624   \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6625   \let\bbl@latinarabic=\@arabic
6626   \let\bbl@0L@@arabic\@arabic
6627   \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
6628   \@ifpackagewith{babel}{bidi=default}%
6629     {\let\bbl@asciroman=\@roman
6630     \let\bbl@0L@@roman\@roman
6631     \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
6632     \let\bbl@asciiRoman=\@Roman
6633     \let\bbl@0L@@roman\@Roman
6634     \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6635     \let\bbl@0L@labelenumii\labelenumii
6636     \def\labelenumii()\theenumii}%
6637     \let\bbl@0L@p@enumiii\p@enumiii
6638     \def\p@enumiii{\p@enumii}\theenumii}}{}{}
6639   <<Footnote changes>>
6640 \IfBabelLayout{footnotes}%
6641   {\let\bbl@0L@footnote\footnote
6642   \BabelFootnote\footnote\languagename{}{}}%
6643   \BabelFootnote\localfootnote\languagename{}{}}%
6644   \BabelFootnote\mainfootnote{}{}}{}
6645 {}

```

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

```

6646 \IfBabelLayout{extras}%
6647   {\bbl@ncarg\let\bbl@0L@underline{underline }%
6648   \bbl@carg\bbl@sreplace{underline }%
6649   {\$@@underline}{\bgroup\bbl@nextfakes$@@underline}%
6650   \bbl@carg\bbl@sreplace{underline }%
6651   {\m@th$}{\m@th$\egroup}%
6652   \let\bbl@0L@LaTeXe\LaTeXe

```

```

6653 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6654 \if b\expandafter\@car\f@series\@nil\boldmath\fi
6655 \babe\sublr{%
6656 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
6657 {}
6658 </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.

```

6659 (*transforms)
6660 Babel.linebreaking.replacements = {}
6661 Babel.linebreaking.replacements[0] = {} -- pre
6662 Babel.linebreaking.replacements[1] = {} -- post
6663
6664 -- Discretionaries contain strings as nodes
6665 function Babel.str_to_nodes(fn, matches, base)
6666 local n, head, last
6667 if fn == nil then return nil end
6668 for s in string.utfvalues(fn(matches)) do
6669   if base.id == 7 then
6670     base = base.replace
6671   end
6672   n = node.copy(base)
6673   n.char = s
6674   if not head then
6675     head = n
6676   else
6677     last.next = n
6678   end
6679   last = n
6680 end
6681 return head
6682 end
6683
6684 Babel.fetch_subtext = {}
6685
6686 Babel.ignore_pre_char = function(node)
6687 return (node.lang == Babel.nohyphenation)
6688 end
6689
6690 -- Merging both functions doesn't seem feasible, because there are too
6691 -- many differences.
6692 Babel.fetch_subtext[0] = function(head)
6693 local word_string = ''
6694 local word_nodes = {}
6695 local lang
6696 local item = head
6697 local inmath = false
6698
6699 while item do
6700
6701   if item.id == 11 then

```



```

6702     inmath = (item.subtype == 0)
6703 end
6704
6705 if inmath then
6706     -- pass
6707
6708 elseif item.id == 29 then
6709     local locale = node.get_attribute(item, Babel.attr_locale)
6710
6711     if lang == locale or lang == nil then
6712         lang = lang or locale
6713         if Babel.ignore_pre_char(item) then
6714             word_string = word_string .. Babel.us_char
6715         else
6716             word_string = word_string .. unicode.utf8.char(item.char)
6717         end
6718         word_nodes[#word_nodes+1] = item
6719     else
6720         break
6721     end
6722
6723 elseif item.id == 12 and item.subtype == 13 then
6724     word_string = word_string .. ' '
6725     word_nodes[#word_nodes+1] = item
6726
6727     -- Ignore leading unrecognized nodes, too.
6728     elseif word_string ~= '' then
6729         word_string = word_string .. Babel.us_char
6730         word_nodes[#word_nodes+1] = item -- Will be ignored
6731     end
6732
6733     item = item.next
6734 end
6735
6736 -- Here and above we remove some trailing chars but not the
6737 -- corresponding nodes. But they aren't accessed.
6738 if word_string:sub(-1) == ' ' then
6739     word_string = word_string:sub(1,-2)
6740 end
6741 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6742 return word_string, word_nodes, item, lang
6743 end
6744
6745 Babel.fetch_subtext[1] = function(head)
6746     local word_string = ''
6747     local word_nodes = {}
6748     local lang
6749     local item = head
6750     local inmath = false
6751
6752     while item do
6753
6754         if item.id == 11 then
6755             inmath = (item.subtype == 0)
6756         end
6757
6758         if inmath then
6759             -- pass
6760
6761         elseif item.id == 29 then
6762             if item.lang == lang or lang == nil then
6763                 if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6764                     lang = lang or item.lang

```

```

6765         word_string = word_string .. unicode.utf8.char(item.char)
6766         word_nodes[#word_nodes+1] = item
6767     end
6768     else
6769         break
6770     end
6771
6772     elseif item.id == 7 and item.subtype == 2 then
6773         word_string = word_string .. '='
6774         word_nodes[#word_nodes+1] = item
6775
6776     elseif item.id == 7 and item.subtype == 3 then
6777         word_string = word_string .. '|'
6778         word_nodes[#word_nodes+1] = item
6779
6780     -- (1) Go to next word if nothing was found, and (2) implicitly
6781     -- remove leading USs.
6782     elseif word_string == '' then
6783         -- pass
6784
6785     -- This is the responsible for splitting by words.
6786     elseif (item.id == 12 and item.subtype == 13) then
6787         break
6788
6789     else
6790         word_string = word_string .. Babel.us_char
6791         word_nodes[#word_nodes+1] = item -- Will be ignored
6792     end
6793
6794     item = item.next
6795 end
6796
6797 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6798 return word_string, word_nodes, item, lang
6799 end
6800
6801 function Babel.pre_hyphenate_replace(head)
6802     Babel.hyphenate_replace(head, 0)
6803 end
6804
6805 function Babel.post_hyphenate_replace(head)
6806     Babel.hyphenate_replace(head, 1)
6807 end
6808
6809 Babel.us_char = string.char(31)
6810
6811 function Babel.hyphenate_replace(head, mode)
6812     local u = unicode.utf8
6813     local lbrk = Babel.linebreaking.replacements[mode]
6814
6815     local word_head = head
6816
6817     while true do -- for each subtext block
6818
6819         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6820
6821         if Babel.debug then
6822             print()
6823             print((mode == 0) and '@@@@<' or '@@@@>', w)
6824         end
6825
6826         if nw == nil and w == '' then break end
6827

```

```

6828     if not lang then goto next end
6829     if not lbkr[lang] then goto next end
6830
6831     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6832     -- loops are nested.
6833     for k=1, #lbkr[lang] do
6834         local p = lbkr[lang][k].pattern
6835         local r = lbkr[lang][k].replace
6836         local attr = lbkr[lang][k].attr or -1
6837
6838         if Babel.debug then
6839             print('*****', p, mode)
6840         end
6841
6842         -- This variable is set in some cases below to the first *byte*
6843         -- after the match, either as found by u.match (faster) or the
6844         -- computed position based on sc if w has changed.
6845         local last_match = 0
6846         local step = 0
6847
6848         -- For every match.
6849         while true do
6850             if Babel.debug then
6851                 print('=====' )
6852             end
6853             local new -- used when inserting and removing nodes
6854
6855             local matches = { u.match(w, p, last_match) }
6856
6857             if #matches < 2 then break end
6858
6859             -- Get and remove empty captures (with ()'s, which return a
6860             -- number with the position), and keep actual captures
6861             -- (from (...)), if any, in matches.
6862             local first = table.remove(matches, 1)
6863             local last = table.remove(matches, #matches)
6864             -- Non re-fetched substrings may contain \31, which separates
6865             -- subsubstrings.
6866             if string.find(w:sub(first, last-1), Babel.us_char) then break end
6867
6868             local save_last = last -- with A()BC()D, points to D
6869
6870             -- Fix offsets, from bytes to unicode. Explained above.
6871             first = u.len(w:sub(1, first-1)) + 1
6872             last = u.len(w:sub(1, last-1)) -- now last points to C
6873
6874             -- This loop stores in a small table the nodes
6875             -- corresponding to the pattern. Used by 'data' to provide a
6876             -- predictable behavior with 'insert' (w_nodes is modified on
6877             -- the fly), and also access to 'remove'd nodes.
6878             local sc = first-1 -- Used below, too
6879             local data_nodes = {}
6880
6881             local enabled = true
6882             for q = 1, last-first+1 do
6883                 data_nodes[q] = w_nodes[sc+q]
6884                 if enabled
6885                     and attr > -1
6886                     and not node.has_attribute(data_nodes[q], attr)
6887                 then
6888                     enabled = false
6889                 end
6890             end

```

```

6891
6892 -- This loop traverses the matched substring and takes the
6893 -- corresponding action stored in the replacement list.
6894 -- sc = the position in substr nodes / string
6895 -- rc = the replacement table index
6896 local rc = 0
6897
6898 while rc < last-first+1 do -- for each replacement
6899     if Babel.debug then
6900         print('.....', rc + 1)
6901     end
6902     sc = sc + 1
6903     rc = rc + 1
6904
6905     if Babel.debug then
6906         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6907         local ss = ''
6908         for itt in node.traverse(head) do
6909             if itt.id == 29 then
6910                 ss = ss .. unicode.utf8.char(itt.char)
6911             else
6912                 ss = ss .. '{' .. itt.id .. '}'
6913             end
6914         end
6915         print('*****', ss)
6916
6917     end
6918
6919     local crep = r[rc]
6920     local item = w_nodes[sc]
6921     local item_base = item
6922     local placeholder = Babel.us_char
6923     local d
6924
6925     if crep and crep.data then
6926         item_base = data_nodes[crep.data]
6927     end
6928
6929     if crep then
6930         step = crep.step or 0
6931     end
6932
6933     if (not enabled) or (crep and next(crep) == nil) then -- = {}
6934         last_match = save_last -- Optimization
6935         goto next
6936
6937     elseif crep == nil or crep.remove then
6938         node.remove(head, item)
6939         table.remove(w_nodes, sc)
6940         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6941         sc = sc - 1 -- Nothing has been inserted.
6942         last_match = utf8.offset(w, sc+1+step)
6943         goto next
6944
6945     elseif crep and crep.kashida then -- Experimental
6946         node.set_attribute(item,
6947             Babel.attr_kashida,
6948             crep.kashida)
6949         last_match = utf8.offset(w, sc+1+step)
6950         goto next
6951
6952     elseif crep and crep.string then
6953         local str = crep.string(matches)

```

```

6954         if str == '' then -- Gather with nil
6955             node.remove(head, item)
6956             table.remove(w_nodes, sc)
6957             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6958             sc = sc - 1 -- Nothing has been inserted.
6959         else
6960             local loop_first = true
6961             for s in string.utfvalues(str) do
6962                 d = node.copy(item_base)
6963                 d.char = s
6964                 if loop_first then
6965                     loop_first = false
6966                     head, new = node.insert_before(head, item, d)
6967                     if sc == 1 then
6968                         word_head = head
6969                     end
6970                     w_nodes[sc] = d
6971                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6972                 else
6973                     sc = sc + 1
6974                     head, new = node.insert_before(head, item, d)
6975                     table.insert(w_nodes, sc, new)
6976                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6977                 end
6978                 if Babel.debug then
6979                     print('.....', 'str')
6980                     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6981                 end
6982             end -- for
6983             node.remove(head, item)
6984         end -- if ''
6985         last_match = utf8.offset(w, sc+1+step)
6986         goto next
6987
6988     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6989         d = node.new(7, 3) -- (disc, regular)
6990         d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
6991         d.post = Babel.str_to_nodes(crep.post, matches, item_base)
6992         d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6993         d.attr = item_base.attr
6994         if crep.pre == nil then -- TeXbook p96
6995             d.penalty = crep.penalty or tex.hyphenpenalty
6996         else
6997             d.penalty = crep.penalty or tex.exhyphenpenalty
6998         end
6999         placeholder = '|'
7000         head, new = node.insert_before(head, item, d)
7001
7002     elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7003         -- ERROR
7004
7005     elseif crep and crep.penalty then
7006         d = node.new(14, 0) -- (penalty, userpenalty)
7007         d.attr = item_base.attr
7008         d.penalty = crep.penalty
7009         head, new = node.insert_before(head, item, d)
7010
7011     elseif crep and crep.space then
7012         -- 655360 = 10 pt = 10 * 65536 sp
7013         d = node.new(12, 13) -- (glue, spaceskip)
7014         local quad = font.getfont(item_base.font).size or 655360
7015         node.setglue(d, crep.space[1] * quad,
7016             crep.space[2] * quad,

```

```

7017             crep.space[3] * quad)
7018     if mode == 0 then
7019         placeholder = ' '
7020     end
7021     head, new = node.insert_before(head, item, d)
7022
7023     elseif crep and crep.spacefactor then
7024         d = node.new(12, 13)      -- (glue, spaceskip)
7025         local base_font = font.getfont(item_base.font)
7026         node.setglue(d,
7027             crep.spacefactor[1] * base_font.parameters['space'],
7028             crep.spacefactor[2] * base_font.parameters['space_stretch'],
7029             crep.spacefactor[3] * base_font.parameters['space_shrink'])
7030         if mode == 0 then
7031             placeholder = ' '
7032         end
7033         head, new = node.insert_before(head, item, d)
7034
7035     elseif mode == 0 and crep and crep.space then
7036         -- ERROR
7037
7038     end -- ie replacement cases
7039
7040     -- Shared by disc, space and penalty.
7041     if sc == 1 then
7042         word_head = head
7043     end
7044     if crep.insert then
7045         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7046         table.insert(w_nodes, sc, new)
7047         last = last + 1
7048     else
7049         w_nodes[sc] = d
7050         node.remove(head, item)
7051         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7052     end
7053
7054     last_match = utf8.offset(w, sc+1+step)
7055
7056     ::next::
7057
7058     end -- for each replacement
7059
7060     if Babel.debug then
7061         print('.....', '/')
7062         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7063     end
7064
7065     end -- for match
7066
7067     end -- for patterns
7068
7069     ::next::
7070     word_head = nw
7071 end -- for substring
7072 return head
7073 end
7074
7075 -- This table stores capture maps, numbered consecutively
7076 Babel.capture_maps = {}
7077
7078 -- The following functions belong to the next macro
7079 function Babel.capture_func(key, cap)

```

```

7080 local ret = "[[" .. cap:gsub('{{[0-9]}}', ")]..m[%1]..["] .. "]"
7081 local cnt
7082 local u = unicode.utf8
7083 ret, cnt = ret:gsub('{{[0-9]}|([^\]|+)|(.-)}', Babel.capture_func_map)
7084 if cnt == 0 then
7085     ret = u.gsub(ret, '{{%x%x%x%x+}}',
7086         function (n)
7087             return u.char(tonumber(n, 16))
7088         end)
7089 end
7090 ret = ret:gsub("%[%[%]]%.", '')
7091 ret = ret:gsub("%.%[%[%]]%", '')
7092 return key .. [=function(m) return ] .. ret .. [= end]]
7093 end
7094
7095 function Babel.capt_map(from, mapno)
7096     return Babel.capture_maps[mapno][from] or from
7097 end
7098
7099 -- Handle the {n|abc|ABC} syntax in captures
7100 function Babel.capture_func_map(capno, from, to)
7101     local u = unicode.utf8
7102     from = u.gsub(from, '{{%x%x%x%x+}}',
7103         function (n)
7104             return u.char(tonumber(n, 16))
7105         end)
7106     to = u.gsub(to, '{{%x%x%x%x+}}',
7107         function (n)
7108             return u.char(tonumber(n, 16))
7109         end)
7110     local froms = {}
7111     for s in string.utfcharacters(from) do
7112         table.insert(froms, s)
7113     end
7114     local cnt = 1
7115     table.insert(Babel.capture_maps, {})
7116     local mlen = table.getn(Babel.capture_maps)
7117     for s in string.utfcharacters(to) do
7118         Babel.capture_maps[mlen][froms[cnt]] = s
7119         cnt = cnt + 1
7120     end
7121     return "]"..Babel.capt_map(m[" .. capno .. "], " ..
7122         (mlen) .. ").." .. "["
7123 end
7124
7125 -- Create/Extend reversed sorted list of kashida weights:
7126 function Babel.capture_kashida(key, wt)
7127     wt = tonumber(wt)
7128     if Babel.kashida_wts then
7129         for p, q in ipairs(Babel.kashida_wts) do
7130             if wt == q then
7131                 break
7132             elseif wt > q then
7133                 table.insert(Babel.kashida_wts, p, wt)
7134                 break
7135             elseif table.getn(Babel.kashida_wts) == p then
7136                 table.insert(Babel.kashida_wts, wt)
7137             end
7138         end
7139     else
7140         Babel.kashida_wts = { wt }
7141     end
7142     return 'kashida = ' .. wt

```

```

7143 end
7144
7145 -- Experimental: applies prehyphenation transforms to a string (letters
7146 -- and spaces).
7147 function Babel.string_prehyphenation(str, locale)
7148   local n, head, last, res
7149   head = node.new(8, 0) -- dummy (hack just to start)
7150   last = head
7151   for s in string.utfvalues(str) do
7152     if s == 20 then
7153       n = node.new(12, 0)
7154     else
7155       n = node.new(29, 0)
7156       n.char = s
7157     end
7158     node.set_attribute(n, Babel.attr_locale, locale)
7159     last.next = n
7160     last = n
7161   end
7162   head = Babel.hyphenate_replace(head, 0)
7163   res = ''
7164   for n in node.traverse(head) do
7165     if n.id == 12 then
7166       res = res .. ' '
7167     elseif n.id == 29 then
7168       res = res .. unicode.utf8.char(n.char)
7169     end
7170   end
7171   tex.print(res)
7172 end
7173 </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):

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

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

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

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

```

7174 (*basic-r)
7175 Babel = Babel or {}
7176
7177 Babel.bidi_enabled = true
7178
7179 require('babel-data-bidi.lua')
7180
7181 local characters = Babel.characters
7182 local ranges = Babel.ranges
7183
7184 local DIR = node.id("dir")
7185
7186 local function dir_mark(head, from, to, outer)
7187   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
7188   local d = node.new(DIR)
7189   d.dir = '+' .. dir
7190   node.insert_before(head, from, d)
7191   d = node.new(DIR)
7192   d.dir = '-' .. dir
7193   node.insert_after(head, to, d)
7194 end
7195
7196 function Babel.bidi(head, ispar)
7197   local first_n, last_n          -- first and last char with nums
7198   local last_es                  -- an auxiliary 'last' used with nums
7199   local first_d, last_d         -- first and last char in L/R block
7200   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):

```

7201   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7202   local strong_lr = (strong == 'l') and 'l' or 'r'
7203   local outer = strong
7204
7205   local new_dir = false
7206   local first_dir = false
7207   local inmath = false
7208
7209   local last_lr
7210
7211   local type_n = ''
7212
7213   for item in node.traverse(head) do
7214
7215     -- three cases: glyph, dir, otherwise
7216     if item.id == node.id'glyph'
7217       or (item.id == 7 and item.subtype == 2) then
7218
7219       local itemchar
7220       if item.id == 7 and item.subtype == 2 then
7221         itemchar = item.replace.char
7222       else
7223         itemchar = item.char
7224       end
7225       local chardata = characters[itemchar]

```

```

7226     dir = chardata and chardata.d or nil
7227     if not dir then
7228         for nn, et in ipairs(ranges) do
7229             if itemchar < et[1] then
7230                 break
7231             elseif itemchar <= et[2] then
7232                 dir = et[3]
7233                 break
7234             end
7235         end
7236     end
7237     dir = dir or 'l'
7238     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).

```

7239     if new_dir then
7240         attr_dir = 0
7241         for at in node.traverse(item.attr) do
7242             if at.number == Babel.attr_dir then
7243                 attr_dir = at.value & 0x3
7244             end
7245         end
7246         if attr_dir == 1 then
7247             strong = 'r'
7248         elseif attr_dir == 2 then
7249             strong = 'al'
7250         else
7251             strong = 'l'
7252         end
7253         strong_lr = (strong == 'l') and 'l' or 'r'
7254         outer = strong_lr
7255         new_dir = false
7256     end
7257
7258     if dir == 'nsm' then dir = strong end           -- W1

```

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

```

7259     dir_real = dir           -- We need dir_real to set strong below
7260     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:

```

7261     if strong == 'al' then
7262         if dir == 'en' then dir = 'an' end           -- W2
7263         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7264         strong_lr = 'r'                               -- W3
7265     end

```

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

```

7266     elseif item.id == node.id'dir' and not inmath then
7267         new_dir = true
7268         dir = nil
7269     elseif item.id == node.id'math' then
7270         inmath = (item.subtype == 0)
7271     else
7272         dir = nil           -- Not a char
7273     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>.

```

7274   if dir == 'en' or dir == 'an' or dir == 'et' then
7275       if dir ~= 'et' then
7276           type_n = dir
7277       end
7278       first_n = first_n or item
7279       last_n = last_es or item
7280       last_es = nil
7281   elseif dir == 'es' and last_n then -- W3+W6
7282       last_es = item
7283   elseif dir == 'cs' then           -- it's right - do nothing
7284   elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7285       if strong_lr == 'r' and type_n ~= '' then
7286           dir_mark(head, first_n, last_n, 'r')
7287       elseif strong_lr == 'l' and first_d and type_n == 'an' then
7288           dir_mark(head, first_n, last_n, 'r')
7289           dir_mark(head, first_d, last_d, outer)
7290           first_d, last_d = nil, nil
7291       elseif strong_lr == 'l' and type_n ~= '' then
7292           last_d = last_n
7293       end
7294       type_n = ''
7295       first_n, last_n = nil, nil
7296   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:

```

7297   if dir == 'l' or dir == 'r' then
7298       if dir ~= outer then
7299           first_d = first_d or item
7300           last_d = item
7301       elseif first_d and dir ~= strong_lr then
7302           dir_mark(head, first_d, last_d, outer)
7303           first_d, last_d = nil, nil
7304       end
7305   end

```

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

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

```

7306   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7307       item.char = characters[item.char] and
7308           characters[item.char].m or item.char
7309   elseif (dir or new_dir) and last_lr ~= item then
7310       local mir = outer .. strong_lr .. (dir or outer)
7311       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7312           for ch in node.traverse(node.next(last_lr)) do
7313               if ch == item then break end
7314               if ch.id == node.id'glyph' and characters[ch.char] then
7315                   ch.char = characters[ch.char].m or ch.char
7316               end
7317           end
7318       end
7319   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).

```

7320   if dir == 'l' or dir == 'r' then
7321       last_lr = item
7322       strong = dir_real           -- Don't search back - best save now
7323       strong_lr = (strong == 'l') and 'l' or 'r'
7324       elseif new_dir then
7325           last_lr = nil
7326       end
7327   end

```

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

```

7328   if last_lr and outer == 'r' then
7329       for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7330           if characters[ch.char] then
7331               ch.char = characters[ch.char].m or ch.char
7332           end
7333       end
7334   end
7335   if first_n then
7336       dir_mark(head, first_n, last_n, outer)
7337   end
7338   if first_d then
7339       dir_mark(head, first_d, last_d, outer)
7340   end

```

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

```

7341   return node.prev(head) or head
7342 end
7343 </basic-r>

```

And here the Lua code for bidi=basic:

```

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

```

```

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

```

```

7440 local last = outer
7441 -- 'al' is only taken into account in the first, current loop
7442 if save_outer == 'al' then save_outer = 'r' end
7443
7444 local fontmap = Babel.fontmap
7445
7446 for item in node.traverse(head) do
7447
7448     -- In what follows, #node is the last (previous) node, because the
7449     -- current one is not added until we start processing the neutrals.
7450
7451     -- three cases: glyph, dir, otherwise
7452     if item.id == GLYPH
7453         or (item.id == 7 and item.subtype == 2) then
7454
7455         local d_font = nil
7456         local item_r
7457         if item.id == 7 and item.subtype == 2 then
7458             item_r = item.replace -- automatic discs have just 1 glyph
7459         else
7460             item_r = item
7461         end
7462         local chardata = characters[item_r.char]
7463         d = chardata and chardata.d or nil
7464         if not d or d == 'nsm' then
7465             for nn, et in ipairs(ranges) do
7466                 if item_r.char < et[1] then
7467                     break
7468                 elseif item_r.char <= et[2] then
7469                     if not d then d = et[3]
7470                     elseif d == 'nsm' then d_font = et[3]
7471                     end
7472                 break
7473             end
7474         end
7475         end
7476         d = d or 'l'
7477
7478         -- A short 'pause' in bidi for mapfont
7479         d_font = d_font or d
7480         d_font = (d_font == 'l' and 0) or
7481                 (d_font == 'nsm' and 0) or
7482                 (d_font == 'r' and 1) or
7483                 (d_font == 'al' and 2) or
7484                 (d_font == 'an' and 2) or nil
7485         if d_font and fontmap and fontmap[d_font][item_r.font] then
7486             item_r.font = fontmap[d_font][item_r.font]
7487         end
7488
7489         if new_d then
7490             table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7491             if inmath then
7492                 attr_d = 0
7493             else
7494                 attr_d = node.get_attribute(item, ATDIR)
7495                 attr_d = attr_d & 0x3
7496             end
7497             if attr_d == 1 then
7498                 outer_first = 'r'
7499                 last = 'r'
7500             elseif attr_d == 2 then
7501                 outer_first = 'r'
7502                 last = 'al'

```

```

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

```

```

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

```



```

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

```

```

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

```

```

7755         d.dir = item.prev.dir
7756         node.remove(head, item.prev)
7757         node.insert_after(head, item, d)
7758     end
7759 end
7760     linking = 0
7761 end
7762 end
7763 end
7764
7765 return head
7766 end
7767 </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.

```

7768 <*nil>
7769 \ProvidesLanguage{nil}[<<date>> v<<version>> Nil language]
7770 \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.

```

7771 \ifx\l@nil\@undefined
7772   \newlanguage\l@nil
7773   \@namedef{bbl@hyphendata@the\l@nil}{}{}% Remove warning
7774   \let\bbl@elt\relax
7775   \edef\bbl@languages{% Add it to the list of languages
7776     \bbl@languages\bbl@elt{nil}{the\l@nil}{}{}}
7777 \fi

```

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

```

7778 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

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

```

\captionnil
\datenil 7779 \let\captionnil\@empty
7780 \let\datenil\@empty

```

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

```

7781 \def\bbl@inidata@nil{%
7782   \bbl@elt{identification}{tag.ini}{und}%
7783   \bbl@elt{identification}{load.level}{0}%

```

```

7784 \bbl@elt{identification}{charset}{utf8}%
7785 \bbl@elt{identification}{version}{1.0}%
7786 \bbl@elt{identification}{date}{2022-05-16}%
7787 \bbl@elt{identification}{name.local}{nil}%
7788 \bbl@elt{identification}{name.english}{nil}%
7789 \bbl@elt{identification}{name.babel}{nil}%
7790 \bbl@elt{identification}{tag.bcp47}{und}%
7791 \bbl@elt{identification}{language.tag.bcp47}{und}%
7792 \bbl@elt{identification}{tag.opentype}{dflt}%
7793 \bbl@elt{identification}{script.name}{Latin}%
7794 \bbl@elt{identification}{script.tag.bcp47}{Latn}%
7795 \bbl@elt{identification}{script.tag.opentype}{DFLT}%
7796 \bbl@elt{identification}{level}{1}%
7797 \bbl@elt{identification}{encodings}{}%
7798 \bbl@elt{identification}{derivate}{no}%
7799 \@namedef{bbl@tbc@nil}{und}
7800 \@namedef{bbl@lbc@nil}{und}
7801 \@namedef{bbl@casing@nil}{und} % TODO
7802 \@namedef{bbl@lotf@nil}{dflt}
7803 \@namedef{bbl@elname@nil}{nil}
7804 \@namedef{bbl@lname@nil}{nil}
7805 \@namedef{bbl@esname@nil}{Latin}
7806 \@namedef{bbl@sname@nil}{Latin}
7807 \@namedef{bbl@sbc@nil}{Latn}
7808 \@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.

```

7809 \ldf@finish{nil}
7810 </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.

```

7811 <<Compute Julian day>> ≡
7812 \def\bbl@fpmo#1#2{(#1-#2*floo(#1/#2))}
7813 \def\bbl@cs@gregleap#1{%
7814   (\bbl@fpmo{#1}{4} == 0) &&
7815   (!( \bbl@fpmo{#1}{100} == 0) && (\bbl@fpmo{#1}{400} != 0))}
7816 \def\bbl@cs@jd#1#2#3{% year, month, day
7817   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
7818     floo((#1 - 1) / 4) + (-floo((#1 - 1) / 100)) +
7819     floo((#1 - 1) / 400) + floo((((367 * #2) - 362) / 12) +
7820     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3)}
7821 <</Compute Julian day>>

```

12.1 Islamic

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

```

7822 <ca-islamic>
7823 \ExplSyntaxOn
7824 <<Compute Julian day>>
7825 % == islamic (default)
7826 % Not yet implemented
7827 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{

```

The Civil calendar.

```

7828 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
7829   ((#3 + ceil(29.5 * (#2 - 1)) +

```

```

7830 (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
7831 1948439.5) - 1) }
7832 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
7833 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
7834 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
7835 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
7836 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
7837 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@@#5#6#7{%
7838 \edef\bbl@tempa{%
7839 \fp_eval:n{ floor(\bbl@cs@jld{#2}{#3}{#4})+0.5 #1}}%
7840 \edef#5{%
7841 \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
7842 \edef#6{\fp_eval:n{
7843 min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
7844 \edef#7{\fp_eval:n{ \bbl@tempa - \bbl@cs@isltojd{#5}{#6}{1} + 1} }}

```

The Umm al-Qura calendar, used mainly in Saudi Arabia, is based on moment-hijri, by Abdullah Alsigar (license MIT).

Since the main aim is to provide a suitable \today, and maybe some close dates, data just covers Hijri ~1435/~1460 (Gregorian ~2014/~2038).

```

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

```

```

7888 \advance\count@\@ne
7889 \ifnum##1>\bbl@tempd\else
7890 \edef\bbl@tempe{\the\count@}%
7891 \edef\bbl@tempb{##1}%
7892 \fi}%
7893 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
7894 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1 ) / 12) }}% annus
7895 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
7896 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
7897 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
7898 \ExplSyntaxOff
7899 \bbl@add\bbl@precalendar{%
7900 \bbl@replace\bbl@ld@calendar{-civil}{}%
7901 \bbl@replace\bbl@ld@calendar{-umalqura}{}%
7902 \bbl@replace\bbl@ld@calendar{+}{}%
7903 \bbl@replace\bbl@ld@calendar{-}{}}
7904 </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 `hebcald.sty`

```

7905 (*ca-hebrew)
7906 \newcount\bbl@cntcommon
7907 \def\bbl@remainder#1#2#3{%
7908 #3=#1\relax
7909 \divide #3 by #2\relax
7910 \multiply #3 by -#2\relax
7911 \advance #3 by #1\relax}%
7912 \newif\ifbbl@divisible
7913 \def\bbl@checkifdivisible#1#2{%
7914 {\countdef\tmp=0
7915 \bbl@remainder{#1}{#2}{\tmp}%
7916 \ifnum \tmp=0
7917 \global\bbl@divisibletrue
7918 \else
7919 \global\bbl@divisiblefalse
7920 \fi}}
7921 \newif\ifbbl@gregleap
7922 \def\bbl@ifgregleap#1{%
7923 \bbl@checkifdivisible{#1}{4}%
7924 \ifbbl@divisible
7925 \bbl@checkifdivisible{#1}{100}%
7926 \ifbbl@divisible
7927 \bbl@checkifdivisible{#1}{400}%
7928 \ifbbl@divisible
7929 \bbl@gregleaptrue
7930 \else
7931 \bbl@gregleapfalse
7932 \fi
7933 \else
7934 \bbl@gregleaptrue
7935 \fi
7936 \else
7937 \bbl@gregleapfalse
7938 \fi
7939 \ifbbl@gregleap}
7940 \def\bbl@gregdayspriormonths#1#2#3{%
7941 {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
7942 181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
7943 \bbl@ifgregleap{#2}%
7944 \ifnum #1 > 2

```

```

7945         \advance #3 by 1
7946     \fi
7947 \fi
7948     \global\bbbl@cntcommon=#3}%
7949     #3=\bbbl@cntcommon}
7950 \def\bbbl@gregdaysprioryears#1#2{%
7951     {\countdef\tmpc=4
7952     \countdef\tmpb=2
7953     \tmpb=#1\relax
7954     \advance \tmpb by -1
7955     \tmpc=\tmpb
7956     \multiply \tmpc by 365
7957     #2=\tmpc
7958     \tmpc=\tmpb
7959     \divide \tmpc by 4
7960     \advance #2 by \tmpc
7961     \tmpc=\tmpb
7962     \divide \tmpc by 100
7963     \advance #2 by -\tmpc
7964     \tmpc=\tmpb
7965     \divide \tmpc by 400
7966     \advance #2 by \tmpc
7967     \global\bbbl@cntcommon=#2\relax}%
7968     #2=\bbbl@cntcommon}
7969 \def\bbbl@absfromgreg#1#2#3#4{%
7970     {\countdef\tmpd=0
7971     #4=#1\relax
7972     \bbbl@gregdayspriormonths{#2}{#3}{\tmpd}%
7973     \advance #4 by \tmpd
7974     \bbbl@gregdaysprioryears{#3}{\tmpd}%
7975     \advance #4 by \tmpd
7976     \global\bbbl@cntcommon=#4\relax}%
7977     #4=\bbbl@cntcommon}
7978 \newif\ifbbbl@hebrleap
7979 \def\bbbl@checkleaphebryear#1{%
7980     {\countdef\tmpa=0
7981     \countdef\tmpb=1
7982     \tmpa=#1\relax
7983     \multiply \tmpa by 7
7984     \advance \tmpa by 1
7985     \bbbl@remainder{\tmpa}{19}{\tmpb}%
7986     \ifnum \tmpb < 7
7987         \global\bbbl@hebrleaptrue
7988     \else
7989         \global\bbbl@hebrleapfalse
7990     \fi}}
7991 \def\bbbl@hebrlapsedmonths#1#2{%
7992     {\countdef\tmpa=0
7993     \countdef\tmpb=1
7994     \countdef\tmpc=2
7995     \tmpa=#1\relax
7996     \advance \tmpa by -1
7997     #2=\tmpa
7998     \divide #2 by 19
7999     \multiply #2 by 235
8000     \bbbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8001     \tmpc=\tmpb
8002     \multiply \tmpb by 12
8003     \advance #2 by \tmpb
8004     \multiply \tmpc by 7
8005     \advance \tmpc by 1
8006     \divide \tmpc by 19
8007     \advance #2 by \tmpc

```

```

8008 \global\bbbl@cntcommon=#2}%
8009 #2=\bbbl@cntcommon}
8010 \def\bbbl@hebreleapseddays#1#2{%
8011 {\countdef\tmpa=0
8012 \countdef\tmpb=1
8013 \countdef\tmpc=2
8014 \bbbl@hebreleapsedmonths{#1}{#2}%
8015 \tmpa=#2\relax
8016 \multiply \tmpa by 13753
8017 \advance \tmpa by 5604
8018 \bbbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8019 \divide \tmpa by 25920
8020 \multiply #2 by 29
8021 \advance #2 by 1
8022 \advance #2 by \tmpa
8023 \bbbl@remainder{#2}{7}{\tmpa}%
8024 \ifnum \tmpc < 19440
8025 \ifnum \tmpc < 9924
8026 \else
8027 \ifnum \tmpa=2
8028 \bbbl@checkleaphebrewyear{#1}% of a common year
8029 \ifbbbl@hebrleap
8030 \else
8031 \advance #2 by 1
8032 \fi
8033 \fi
8034 \fi
8035 \ifnum \tmpc < 16789
8036 \else
8037 \ifnum \tmpa=1
8038 \advance #1 by -1
8039 \bbbl@checkleaphebrewyear{#1}% at the end of leap year
8040 \ifbbbl@hebrleap
8041 \advance #2 by 1
8042 \fi
8043 \fi
8044 \fi
8045 \else
8046 \advance #2 by 1
8047 \fi
8048 \bbbl@remainder{#2}{7}{\tmpa}%
8049 \ifnum \tmpa=0
8050 \advance #2 by 1
8051 \else
8052 \ifnum \tmpa=3
8053 \advance #2 by 1
8054 \else
8055 \ifnum \tmpa=5
8056 \advance #2 by 1
8057 \fi
8058 \fi
8059 \fi
8060 \global\bbbl@cntcommon=#2\relax}%
8061 #2=\bbbl@cntcommon}
8062 \def\bbbl@daysinhebrewyear#1#2{%
8063 {\countdef\tmpe=12
8064 \bbbl@hebreleapseddays{#1}{\tmpe}%
8065 \advance #1 by 1
8066 \bbbl@hebreleapseddays{#1}{#2}%
8067 \advance #2 by -\tmpe
8068 \global\bbbl@cntcommon=#2}%
8069 #2=\bbbl@cntcommon}
8070 \def\bbbl@hebrdayspriormonths#1#2#3{%

```



```

8071 {\countdef\tmpf= 14
8072 #3=\ifcase #1\relax
8073     0 \or
8074     0 \or
8075     30 \or
8076     59 \or
8077     89 \or
8078     118 \or
8079     148 \or
8080     148 \or
8081     177 \or
8082     207 \or
8083     236 \or
8084     266 \or
8085     295 \or
8086     325 \or
8087     400
8088 \fi
8089 \bbl@checkleaphebrewyear{#2}%
8090 \ifbbl@hebrleap
8091     \ifnum #1 > 6
8092         \advance #3 by 30
8093     \fi
8094 \fi
8095 \bbl@daysinhebrewyear{#2}{\tmpf}%
8096 \ifnum #1 > 3
8097     \ifnum \tmpf=353
8098         \advance #3 by -1
8099     \fi
8100     \ifnum \tmpf=383
8101         \advance #3 by -1
8102     \fi
8103 \fi
8104 \ifnum #1 > 2
8105     \ifnum \tmpf=355
8106         \advance #3 by 1
8107     \fi
8108     \ifnum \tmpf=385
8109         \advance #3 by 1
8110     \fi
8111 \fi
8112 \global\bbl@cntcommon=#3\relax}%
8113 #3=\bbl@cntcommon}
8114 \def\bbl@absfromhebr#1#2#3#4{%
8115     {#4=#1\relax
8116     \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8117     \advance #4 by #1\relax
8118     \bbl@hebreleaseddays{#3}{#1}%
8119     \advance #4 by #1\relax
8120     \advance #4 by -1373429
8121     \global\bbl@cntcommon=#4\relax}%
8122 #4=\bbl@cntcommon}
8123 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8124     {\countdef\tmpx= 17
8125     \countdef\tmpy= 18
8126     \countdef\tmpz= 19
8127     #6=#3\relax
8128     \global\advance #6 by 3761
8129     \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8130     \tmpz=1 \tmpy=1
8131     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8132     \ifnum \tmpx > #4\relax
8133         \global\advance #6 by -1

```

```

8134     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8135     \fi
8136     \advance #4 by -\tmpx
8137     \advance #4 by 1
8138     #5=#4\relax
8139     \divide #5 by 30
8140     \loop
8141         \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8142         \ifnum \tmpx < #4\relax
8143             \advance #5 by 1
8144             \tmpy=\tmpx
8145         \repeat
8146     \global\advance #5 by -1
8147     \global\advance #4 by -\tmpy}}
8148 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
8149 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8150 \def\bbl@ca@hebrew#1-#2-#3\@#4#5#6{%
8151     \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8152     \bbl@hebrfromgreg
8153     {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8154     {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
8155     \edef#4{\the\bbl@hebryear}%
8156     \edef#5{\the\bbl@hebrmonth}%
8157     \edef#6{\the\bbl@hebrday}}
8158 </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).

```

8159 (*ca-persian)
8160 \ExplSyntaxOn
8161 <<Compute Julian day>>
8162 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8163     2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8164 \def\bbl@ca@persian#1-#2-#3\@#4#5#6{%
8165     \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8166     \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8167         \bbl@afterfi\expandafter\gobble
8168     \fi\fi
8169     {\bbl@error{Year~out~of~range}{The~allowed~range~is~2013-2050}}%
8170     \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8171     \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8172     \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8173     \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8174     \ifnum\bbl@tempc<\bbl@tempb
8175         \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8176         \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8177         \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8178         \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8179     \fi
8180     \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8181     \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8182     \edef#5{\fp_eval:n{% set Jalali month
8183         (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8184     \edef#6{\fp_eval:n{% set Jalali day
8185         (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6))}}}}
8186 \ExplSyntaxOff
8187 </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.

```
8188 (*ca-coptic)
8189 \ExplSyntaxOn
8190 <<Compute Julian day>>
8191 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
8192   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8193   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8194   \edef#4{\fp_eval:n{%
8195     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8196   \edef\bbl@tempc{\fp_eval:n{%
8197     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8198   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8199   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}
8200 \ExplSyntaxOff
8201 </ca-coptic>
8202 (*ca-ethiopic)
8203 \ExplSyntaxOn
8204 <<Compute Julian day>>
8205 \def\bbl@ca@ethiopic#1-#2-#3\@#4#5#6{%
8206   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8207   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8208   \edef#4{\fp_eval:n{%
8209     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8210   \edef\bbl@tempc{\fp_eval:n{%
8211     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8212   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8213   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}
8214 \ExplSyntaxOff
8215 </ca-ethiopic>
```

12.5 Buddhist

That's very simple.

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

```

8243     \count@\@ne
8244     \advance\@tempcnta\@ne\fi
8245     \bbl@xin@{,##1,}{,\bbl@cs@chinese@leap,}%
8246     \ifin@
8247     \advance\count@\m@ne
8248     \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
8249     \else
8250     \edef\bbl@tempe{\the\count@}%
8251     \fi
8252     \edef\bbl@tempb{##1}%
8253     \fi}%
8254 \edef#4{\the\@tempcnta}%
8255 \edef#5{\bbl@tempe}%
8256 \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
8257 \def\bbl@cs@chinese@leap{%
8258 885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
8259 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
8260 354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
8261 768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
8262 1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
8263 1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
8264 1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
8265 2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
8266 2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
8267 2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
8268 3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
8269 3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
8270 3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
8271 4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
8272 4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
8273 5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
8274 5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
8275 5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
8276 6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
8277 6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
8278 6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
8279 7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
8280 7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
8281 7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
8282 8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8283 8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8284 8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
8285 9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
8286 9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
8287 10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
8288 10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
8289 10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
8290 10896,10926,10956,10986,11015,11045,11074,11103}
8291 \ExplSyntaxOff
8292 </ca-chinese>

```

13 Support for Plain T_EX (plain.def)

13.1 Not renaming hyphen.tex

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

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

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

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

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

```
8293 <{*bplain | blplain)
8294 \catcode`\{=1 % left brace is begin-group character
8295 \catcode`\}=2 % right brace is end-group character
8296 \catcode`\#=6 % hash mark is macro parameter character
```

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

```
8297 \openin 0 hyphen.cfg
8298 \ifeof0
8299 \else
8300 \let\a\input
```

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

```
8301 \def\input #1 {%
8302   \let\input\a
8303   \a hyphen.cfg
8304   \let\a\undefined
8305 }
8306 \fi
8307 </bplain | blplain)
```

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

```
8308 (bplain)\a plain.tex
8309 (blplain)\a lplain.tex
```

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

```
8310 (bplain)\def\fmtname{babel-plain}
8311 (blplain)\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 \TeX features

The file `babel.def` expects some definitions made in the $\TeX 2_{\epsilon}$ 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).

```
8312 <<{*Emulate LaTeX}> ≡
8313 \def\@empty{}
8314 \def\loadlocalcfg#1{%
8315   \openin0#1.cfg
8316   \ifeof0
8317     \closein0
8318   \else
8319     \closein0
8320     {\immediate\writel6{*****}%
8321      \immediate\writel6{* Local config file #1.cfg used}%
8322      \immediate\writel6{*}%
8323     }
```

```

8324 \input #1.cfg\relax
8325 \fi
8326 \@endofldf}

```

13.3 General tools

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

```

8327 \long\def\@firstofone#1{#1}
8328 \long\def\@firstoftwo#1#2{#1}
8329 \long\def\@secondoftwo#1#2{#2}
8330 \def\@nnil{\@nil}
8331 \def\@gobbletwo#1#2{}
8332 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
8333 \def\@star@or@long#1{%
8334 \@ifstar
8335 {\let\l@ngrel@x\relax#1}%
8336 {\let\l@ngrel@x\long#1}}
8337 \let\l@ngrel@x\relax
8338 \def\@car#1#2\@nil{#1}
8339 \def\@cdr#1#2\@nil{#2}
8340 \let\@typeset@protect\relax
8341 \let\protected@edef\edef
8342 \long\def\@gobble#1{}
8343 \edef\@backslashchar{\expandafter\@gobble\string\}
8344 \def\strip@prefix#1>{}
8345 \def@g@addto@macro#1#2{%
8346 \toks@\expandafter{#1#2}%
8347 \xdef#1{\the\toks@}}
8348 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8349 \def\@nameuse#1{\csname #1\endcsname}
8350 \def\@ifundefined#1{%
8351 \expandafter\ifx\csname#1\endcsname\relax
8352 \expandafter\@firstoftwo
8353 \else
8354 \expandafter\@secondoftwo
8355 \fi}
8356 \def\@expandtwoargs#1#2#3{%
8357 \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8358 \def\zap@space#1 #2{%
8359 #1%
8360 \ifx#2\@empty\else\expandafter\zap@space\fi
8361 #2}
8362 \let\bbl@trace\@gobble
8363 \def\bbl@error#1#2{%
8364 \begingroup
8365 \newlinechar=\^^J
8366 \def\{\^^J(babel) }%
8367 \errhelp{#2}\errmessage{\#\#1}%
8368 \endgroup}
8369 \def\bbl@warning#1{%
8370 \begingroup
8371 \newlinechar=\^^J
8372 \def\{\^^J(babel) }%
8373 \message{\#\#1}%
8374 \endgroup}
8375 \let\bbl@infowarn\bbl@warning
8376 \def\bbl@info#1{%
8377 \begingroup
8378 \newlinechar=\^^J
8379 \def\{\^^J}%
8380 \wlog{#1}%
8381 \endgroup}

```

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

```
8382 \ifx\@preamblecmds\undefined
8383   \def\@preamblecmds{}
8384 \fi
8385 \def\@onlypreamble#1{%
8386   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8387     \@preamblecmds\do#1}}
8388 \@onlypreamble\@onlypreamble
```

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

```
8389 \def\begindocument{%
8390   \@begindocumenthook
8391   \global\let\@begindocumenthook\undefined
8392   \def\do##1{\global\let##1\undefined}%
8393   \@preamblecmds
8394   \global\let\do\noexpand}

8395 \ifx\@begindocumenthook\undefined
8396   \def\@begindocumenthook{}
8397 \fi
8398 \@onlypreamble\@begindocumenthook
8399 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}
```

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

```
8400 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
8401 \@onlypreamble\AtEndOfPackage
8402 \def\@endofldf{}
8403 \@onlypreamble\@endofldf
8404 \let\bbl@afterlang\@empty
8405 \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.

```
8406 \catcode`\&=\z@
8407 \ifx&\if@filesw\undefined
8408   \expandafter\let\csname if@filesw\expandafter\endcsname
8409     \csname iffalse\endcsname
8410 \fi
8411 \catcode`\&=4
```

Mimick \LaTeX 's commands to define control sequences.

```
8412 \def\newcommand{\@star@or@long\new@command}
8413 \def\new@command#1{%
8414   \@testopt{\@newcommand#1}0}
8415 \def\@newcommand#1[#2]{%
8416   \@ifnextchar [{\@xargdef#1[#2]}%
8417     {\@argdef#1[#2]}}
8418 \long\def\@argdef#1[#2]#3{%
8419   \@yargdef#1\@ne{#2}{#3}}
8420 \long\def\@xargdef#1[#2][#3]#4{%
8421   \expandafter\def\expandafter#1\expandafter{%
8422     \expandafter\@protected@testopt\expandafter #1%
8423     \csname\string#1\expandafter\endcsname{#3}}%
8424   \expandafter\@yargdef\@csname\string#1\endcsname
8425     \tw@{#2}{#4}}
8426 \long\def\@yargdef#1#2#3{%
8427   \@tempcnta#3\relax
8428   \advance \@tempcnta \@ne
8429   \let\@hash@\relax
8430   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
8431   \@tempcntb #2%
```

```

8432 \@whilenum\@tempcntb <\@tempcnta
8433 \do{%
8434   \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
8435   \advance\@tempcntb \@ne}%
8436   \let\@hash@###
8437   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
8438 \def\providecommand{\@star@or@long\provide@command}
8439 \def\provide@command#1{%
8440   \begingroup
8441     \escapechar@m@ne\xdef\@gtempa{\string#1}%
8442   \endgroup
8443   \expandafter\ifundefined\@gtempa
8444     {\def\reserved@a{\new@command#1}}%
8445     {\let\reserved@a\relax
8446     \def\reserved@a{\new@command\reserved@a}}%
8447   \reserved@a}%

8448 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
8449 \def\declare@robustcommand#1{%
8450   \edef\reserved@a{\string#1}%
8451   \def\reserved@b{#1}%
8452   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
8453   \edef#1{%
8454     \ifx\reserved@a\reserved@b
8455       \noexpand\x@protect
8456       \noexpand#1%
8457     \fi
8458     \noexpand\protect
8459     \expandafter\noexpand\csname
8460       \expandafter\@gobble\string#1 \endcsname
8461   }%
8462   \expandafter\new@command\csname
8463     \expandafter\@gobble\string#1 \endcsname
8464 }
8465 \def\x@protect#1{%
8466   \ifx\protect\@typeset@protect\else
8467     \@x@protect#1%
8468   \fi
8469 }
8470 \catcode`\&=\z@ % Trick to hide conditionals
8471 \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`.

```

8472 \def\bbl@tempa{\csname newif\endcsname&fiin@}
8473 \catcode`\&=4
8474 \ifx\in@\@undefined
8475   \def\in@#1#2{%
8476     \def\in@##1#1##2##3\in@{%
8477       \ifx\in@##2\in@false\else\in@true\fi}%
8478     \in@#2#1\in@\in@@}
8479 \else
8480   \let\bbl@tempa\@empty
8481 \fi
8482 \bbl@tempa

```

\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 \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).

```

8483 \def\@ifpackagewith#1#2#3#4{#3}

```


The \LaTeX macro `\ifloaded` checks whether a file was loaded. This functionality is not needed for plain \TeX but we need the macro to be defined as a no-op.

```
8484 \def\ifloaded#1#2#3#4{}
```

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

```
8485 \ifx\@tempcnta\undefined
8486   \csname newcount\endcsname\@tempcnta\relax
8487 \fi
8488 \ifx\@tempcntb\undefined
8489   \csname newcount\endcsname\@tempcntb\relax
8490 \fi
```

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

```
8491 \ifx\bye\undefined
8492   \advance\count10 by -2\relax
8493 \fi
8494 \ifx\ifnextchar\undefined
8495   \def\ifnextchar#1#2#3{%
8496     \let\reserved@#1%
8497     \def\reserved@a{#2}\def\reserved@b{#3}%
8498     \futurelet\@let@token\ifnch}
8499   \def\ifnch{%
8500     \ifx\@let@token\@sptoken
8501       \let\reserved@c\@xifnch
8502     \else
8503       \ifx\@let@token\reserved@d
8504         \let\reserved@c\reserved@a
8505       \else
8506         \let\reserved@c\reserved@b
8507     \fi
8508   \fi
8509   \reserved@c}
8510   \def\{\let\@sptoken= } \: % this makes \@sptoken a space token
8511   \def\{\@xifnch} \expandafter\def\{\futurelet\@let@token\ifnch}
8512 \fi
8513 \def\@testopt#1#2{%
8514   \ifnextchar[#{#1}#{1}[#2]}
8515 \def\@protected@testopt#1{%
8516   \ifx\protect\@typeset@protect
8517     \expandafter\@testopt
8518   \else
8519     \@x@protect#1%
8520   \fi}
8521 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@whilenum{#1\relax
8522   #2\relax}\fi}
8523 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8524   \else\expandafter\@gobble\fi{#1}}
```

13.4 Encoding related macros

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

```
8525 \def\DeclareTextCommand{%
8526   \@dec@text@cmd\providecommand
8527 }
8528 \def\ProvideTextCommand{%
8529   \@dec@text@cmd\providecommand
8530 }
8531 \def\DeclareTextSymbol#1#2#3{%
8532   \@dec@text@cmd\chardef#1{#2}#3\relax
8533 }
```

```

8534 \def\@dec@text@cmd#1#2#3{%
8535   \expandafter\def\expandafter#2%
8536     \expandafter{%
8537       \csname#3-cmd\expandafter\endcsname
8538       \expandafter#2%
8539       \csname#3\string#2\endcsname
8540     }%
8541 %   \let\@ifdefinable\@rc@ifdefinable
8542 \expandafter#1\csname#3\string#2\endcsname
8543 }
8544 \def\@current@cmd#1{%
8545   \ifx\protect\@typeset@protect\else
8546     \noexpand#1\expandafter\@gobble
8547   \fi
8548 }
8549 \def\@changed@cmd#1#2{%
8550   \ifx\protect\@typeset@protect
8551     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8552       \expandafter\ifx\csname ?\string#1\endcsname\relax
8553         \expandafter\def\csname ?\string#1\endcsname{%
8554           \@changed@x@err{#1}%
8555         }%
8556       \fi
8557       \global\expandafter\let
8558         \csname\cf@encoding ?\string#1\endcsname
8559         \csname ?\string#1\endcsname
8560       \fi
8561       \csname\cf@encoding\string#1%
8562       \expandafter\endcsname
8563     \else
8564       \noexpand#1%
8565     \fi
8566 }
8567 \def\@changed@x@err#1{%
8568   \errhelp{Your command will be ignored, type <return> to proceed}%
8569   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}%
8570 \def\DeclareTextCommandDefault#1{%
8571   \DeclareTextCommand#1?%
8572 }
8573 \def\ProvideTextCommandDefault#1{%
8574   \ProvideTextCommand#1?%
8575 }
8576 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8577 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8578 \def\DeclareTextAccent#1#2#3{%
8579   \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
8580 }
8581 \def\DeclareTextCompositeCommand#1#2#3#4{%
8582   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8583   \edef\reserved@b{\string##1}%
8584   \edef\reserved@c{%
8585     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8586   \ifx\reserved@b\reserved@c
8587     \expandafter\expandafter\expandafter\ifx
8588       \expandafter\@car\reserved@a\relax\relax\@nil
8589     \@text@composite
8590   \else
8591     \edef\reserved@b##1{%
8592       \def\expandafter\noexpand
8593       \csname#2\string#1\endcsname###1{%
8594         \noexpand\@text@composite
8595         \expandafter\noexpand\csname#2\string#1\endcsname
8596         ###1\noexpand\@empty\noexpand\@text@composite

```

```

8597         {##1}%
8598     }%
8599 }%
8600 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8601 \fi
8602 \expandafter\def\csname\expandafter\string\csname
8603     #2\endcsname\string#1-\string#3\endcsname{#4}
8604 \else
8605     \errhelp{Your command will be ignored, type <return> to proceed}%
8606     \errmessage{\string\DeclareTextCompositeCommand\space used on
8607         inappropriate command \protect#1}
8608 \fi
8609 }
8610 \def\@text@composite#1#2#3\@text@composite{%
8611     \expandafter\@text@composite@x
8612     \csname\string#1-\string#2\endcsname
8613 }
8614 \def\@text@composite@x#1#2{%
8615     \ifx#1\relax
8616         #2%
8617     \else
8618         #1%
8619     \fi
8620 }
8621 %
8622 \def\@strip@args#1:#2-#3\@strip@args{#2}
8623 \def\DeclareTextComposite#1#2#3#4{%
8624     \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8625     \bgroup
8626         \lccode`\@=#4%
8627         \lowercase{%
8628     \egroup
8629         \reserved@a @%
8630     }%
8631 }
8632 %
8633 \def\UseTextSymbol#1#2{#2}
8634 \def\UseTextAccent#1#2#3{#3}
8635 \def\@use@text@encoding#1{#1}
8636 \def\DeclareTextSymbolDefault#1#2{%
8637     \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8638 }
8639 \def\DeclareTextAccentDefault#1#2{%
8640     \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8641 }
8642 \def\cf@encoding{OT1}

```

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

```

8643 \DeclareTextAccent{"}{OT1}{127}
8644 \DeclareTextAccent{\'}{OT1}{19}
8645 \DeclareTextAccent{\^}{OT1}{94}
8646 \DeclareTextAccent{\`}{OT1}{18}
8647 \DeclareTextAccent{\~}{OT1}{126}

```

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

```

8648 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8649 \DeclareTextSymbol{\textquotedblright}{OT1}{`\'}
8650 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
8651 \DeclareTextSymbol{\textquoteright}{OT1}{``'}
8652 \DeclareTextSymbol{\i}{OT1}{16}
8653 \DeclareTextSymbol{\ss}{OT1}{25}

```

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

```
8654 \ifx\scriptsize\@undefined
8655   \let\scriptsize\sevenrm
8656 \fi
```

And a few more “dummy” definitions.

```
8657 \def\languagename{english}%
8658 \let\bbl@opt@shorthands\@nnil
8659 \def\bbl@ifshorthand#1#2#3{#2}%
8660 \let\bbl@language@opts\@empty
8661 \let\bbl@ensureinfo\@gobble
8662 \let\bbl@provide@locale\relax
8663 \ifx\babeloptionstrings\@undefined
8664   \let\bbl@opt@strings\@nnil
8665 \else
8666   \let\bbl@opt@strings\babeloptionstrings
8667 \fi
8668 \def\BabelStringsDefault{generic}
8669 \def\bbl@tempa{normal}
8670 \ifx\babeloptionmath\bbl@tempa
8671   \def\bbl@mathnormal{\noexpand\textormath}
8672 \fi
8673 \def\AfterBabelLanguage#1#2{}
8674 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
8675 \let\bbl@afterlang\relax
8676 \def\bbl@opt@safe{BR}
8677 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
8678 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
8679 \expandafter\newif\csname ifbbl@single\endcsname
8680 \chardef\bbl@bidimode\z@
8681 <</Emulate LaTeX>>
```

A proxy file:

```
8682 <*plain>
8683 \input babel.def
8684 </plain>
```

14 Acknowledgements

I would like to thank all who volunteered as β -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

There are also many contributors for specific languages, which are mentioned in the respective files. Without them, babel just wouldn't exist.

References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national \LaTeX styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The \TeX book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport, *\LaTeX , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in: \TeX hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.

- [9] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018
- [10] Hubert Partl, *German T_EX*, *TUGboat* 9 (1988) #1, p. 70–72.
- [11] Joachim Schrod, *International L^AT_EX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using L^AT_EX*, Springer, 2002, p. 301–373.
- [13] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).