

Babel

Code

Version 3.98
2023/12/06

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Localization and
internationalization

Unicode

TEX

pdfTEX

LuaTEX

XeTEX

Contents

1	Identification and loading of required files	3
2	locale directory	3
3	Tools	3
3.1	Multiple languages	7
3.2	The Package File (\LaTeX , <code>babel.sty</code>)	8
3.3	<code>base</code>	9
3.4	<code>key=value</code> options and other general option	10
3.5	Conditional loading of shorthands	11
3.6	Interlude for Plain	13
4	Multiple languages	13
4.1	Selecting the language	15
4.2	Errors	23
4.3	Hooks	25
4.4	Setting up language files	27
4.5	Shorthands	29
4.6	Language attributes	38
4.7	Support for saving macro definitions	40
4.8	Short tags	41
4.9	Hyphens	42
4.10	Multiencoding strings	43
4.11	Macros common to a number of languages	49
4.12	Making glyphs available	49
4.12.1	Quotation marks	49
4.12.2	Letters	50
4.12.3	Shorthands for quotation marks	51
4.12.4	Umlauts and tremas	52
4.13	Layout	53
4.14	Load engine specific macros	54
4.15	Creating and modifying languages	54
5	Adjusting the Babel behavior	77
5.1	Cross referencing macros	79
5.2	Marks	82
5.3	Preventing clashes with other packages	83
5.3.1	<code>ifthen</code>	83
5.3.2	<code>varioref</code>	84
5.3.3	<code>hhline</code>	84
5.4	Encoding and fonts	85
5.5	Basic bidi support	86
5.6	Local Language Configuration	90
5.7	Language options	90
6	The kernel of Babel (<code>babel.def</code>, <code>common</code>)	93
7	Loading hyphenation patterns	94
8	Font handling with <code>fontspec</code>	98
9	Hooks for XeTeX and LuaTeX	101
9.1	XeTeX	101

10	Support for interchar	103
10.1	Layout	105
10.2	8-bit TeX	107
10.3	LuaTeX	107
10.4	Southeast Asian scripts	113
10.5	CJK line breaking	115
10.6	Arabic justification	117
10.7	Common stuff	121
10.8	Automatic fonts and ids switching	121
10.9	Bidi	128
10.10	Layout	130
10.11	Lua: transforms	137
10.12	Lua: Auto bidi with basic and basic-r	146
11	Data for CJK	156
12	The ‘nil’ language	157
13	Calendars	158
13.1	Islamic	158
13.2	Hebrew	159
13.3	Persian	163
13.4	Coptic and Ethiopic	164
13.5	Buddhist	165
14	Support for Plain TeX (plain.def)	166
14.1	Not renaming hyphen.tex	166
14.2	Emulating some L ^A T _E X features	167
14.3	General tools	167
14.4	Encoding related macros	171
15	Acknowledgements	174

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

1 Identification and loading of required files

Code documentation is still under revision.

The babel package after unpacking consists of the following files:

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

babel.def is loaded by Plain.

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

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

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

There some additional tex, def and lua files

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

2 locale directory

A required component of babel is a set of ini files with basic definitions for about 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=3.98>>
2 <<date=2023/12/06>>
```

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

```

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

```

76 \def\bb@ifblank#1{%
77   \bb@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
78 \long\def\bb@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
79 \def\bb@ifset#1#2#3{%
80   \bb@ifunset{#1}{#3}{\bb@exp{\@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 checking 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     \beginingroup
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   \beginingroup
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\bb@bsphack{%
163 \ifhmode
164 \hskip\z@skip
165 \def\bb@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
166 \else
167 \let\bb@esphack\@empty
168 \fi}

```

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

```

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

```

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

```

181 \def\bb@extras@wrap#1#2#3{% 1:in-test, 2:before, 3:after
182 \toks@\expandafter\expandafter\expandafter{%
183 \csname extras\language\endcsname}%
184 \bb@exp{\in@{#1}}{\the\toks@}}%
185 \ifin@\else
186 \@temptokena{#2}%
187 \edef\bb@tempc{\the\@temptokena\the\toks@}%
188 \toks@\expandafter{\bb@tempc#3}%
189 \expandafter\edef\csname extras\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 \LaTeX . After it, we will resume the \LaTeX -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{#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@initoload\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\languagename\@undefined\else
577     \ifx\currentgrouplevel\@undefined
578       \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
579     \else
580       \ifnum\currentgrouplevel=\z@
581         \xdef\bbl@language@stack{\languagename+}%
582       \else
583         \xdef\bbl@language@stack{\languagename+\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 `\languagename`. For this we first define a helper function.

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

```
587 \def\bbl@pop@lang#1+#2\@{%
588   \edef\languagename{#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{\languagename}%
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@\languagename}%
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 `\language`. 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 hmap
667     \fi
668     \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
669     % set name
670     \edef\languagename{#1}%
671     \bbl@fixname\languagename
672     % TODO. name@map must be here?
673     \bbl@provide@locale
674     \bbl@iflanguage\languagename{%
675         \let\bbl@select@type\z@
676         \expandafter\bbl@switch\expandafter{\languagename}}
677 \def\babel@aux#1#2{%
678     \select@language{#1}%
679     \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
680         \writefile{##1}{\babel@toc{#1}{#2}\relax}}% TODO - plain?
681 \def\babel@toc#1#2{%
682     \select@language{#1}}

```

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

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

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

The switching of the values of `\lefthyphenmin` and `\righthyphenmin` is somewhat different. First we save their current values, then we check if `\langhyphenmins` is defined. If it is not, we set default values (2 and 3), otherwise the values in `\langhyphenmins` 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\bb@beforeforeign{
787 \edef\foreignlanguage{%
788 \noexpand\protect
789 \expandafter\noexpand\csname foreignlanguage \endcsname}
790 \expandafter\def\csname foreignlanguage \endcsname{%
791 \@ifstar\bb@foreign@s\bb@foreign@x}
792 \providecommand\bb@foreign@x[3][]{%
793 \begingroup
794 \def\bb@selectorname{foreign}%
795 \def\bb@select@opts{#1}%
796 \let\BabelText\@firstofone
797 \bb@beforeforeign
798 \foreign@language{#2}%
799 \bb@usehooks{foreign}{}%
800 \BabelText{#3}% Now in horizontal mode!
801 \endgroup}
802 \def\bb@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
803 \begingroup
804 {\par}%
805 \def\bb@selectorname{foreign*}%
806 \let\bb@select@opts\@empty
807 \let\BabelText\@firstofone
808 \foreign@language{#1}%
809 \bb@usehooks{foreign*}{}%
810 \bb@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 `bb@switch`.

```

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

```

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

```

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

```

`\bb@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). `\bb@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@hymapset=\@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\bbl@withactive#1#2{%
988   \begingroup
989   \lccode`~=#2\relax
990   \lowercase{\endgroup#1~}}

```

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

```

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

```

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

```

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

```

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

```

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

```

4.3 Hooks

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

```

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

```

```

1012 \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1013 \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1014 \bbl@ifunset{\bbl@ev@#2@#3@#1}%
1015   {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1016   {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1017 \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1018 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1019 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1020 \def\bbl@usehooks{\bbl@usehooks@lang\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 `\protect`s the active character whenever `\protect` is *not* `\@typeset@protect`. The `\@gobble` is needed to remove a token such as `\activechar:` (when the double colon was the active character to be dealt with). There are two definitions, depending of `\ifincsname` is available. If there is, the expansion will be more robust.

```

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 latter 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}{%
```

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

```

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\the\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)”. `\nobreak` 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\nobreak#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\nullhyphen
1746   \else
1747     \char\hyphenchar\font
1748   \fi}
```

Finally, we define the hyphen “types”. Their names will not change, so you may use them in ldf’s. After a space, the `\mbox` in `\bb@hy@nobreak` 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@nobreak{\bb@usehyphen{\mbox{\bb@hyphenchar}}}
1754 \def\bb@hy@nobreak{\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 following option is currently no-op. It was meant for the deprecated `\SetCase`.

```
1766 <<{*More package options}>> ≡
1767 \DeclareOption{nocase}{}
1768 <</More package options>>
```

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

```
1769 <<{*More package options}>> ≡
1770 \let\bb@opt@strings\@nnil % accept strings=value
1771 \DeclareOption{strings}{\def\bb@opt@strings{\BabelStringsDefault}}
1772 \DeclareOption{strings=encoded}{\let\bb@opt@strings\relax}
1773 \def\BabelStringsDefault{generic}
1774 <</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.

```

1775 \@onlypreamble\StartBabelCommands
1776 \def\StartBabelCommands{%
1777   \begingroup
1778   \@tempcnta="7F
1779   \def\bbbl@tempa{%
1780     \ifnum\@tempcnta>"FF\else
1781       \catcode\@tempcnta=11
1782       \advance\@tempcnta\@ne
1783       \expandafter\bbbl@tempa
1784       \fi}%
1785   \bbbl@tempa
1786   <<Macros local to BabelCommands>>
1787   \def\bbbl@provstring##1##2{%
1788     \providecommand##1{##2}%
1789     \bbbl@toglobal##1}%
1790   \global\let\bbbl@scafter\@empty
1791   \let\StartBabelCommands\bbbl@startcmds
1792   \ifx\BabelLanguages\relax
1793     \let\BabelLanguages\CurrentOption
1794   \fi
1795   \begingroup
1796   \let\bbbl@screset\@nnil % local flag - disable 1st stopcommands
1797   \StartBabelCommands}
1798 \def\bbbl@startcmds{%
1799   \ifx\bbbl@screset\@nnil\else
1800     \bbbl@usehooks{stopcommands}{}%
1801   \fi
1802   \endgroup
1803   \begingroup
1804   \@ifstar
1805     {\ifx\bbbl@opt@strings\@nnil
1806       \let\bbbl@opt@strings\BabelStringsDefault
1807     \fi
1808     \bbbl@startcmds@i}%
1809   \bbbl@startcmds@i}
1810 \def\bbbl@startcmds@i#1#2{%
1811   \edef\bbbl@L{\zap@space#1 \@empty}%
1812   \edef\bbbl@G{\zap@space#2 \@empty}%
1813   \bbbl@startcmds@ii}
1814 \let\bbbl@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.

```

1815 \newcommand\bbbl@startcmds@ii[1][\@empty]{%
1816   \let\SetString\@gobbletwo
1817   \let\bbbl@stringdef\@gobbletwo
1818   \let\AfterBabelCommands\@gobble
1819   \ifx\@empty#1%
1820     \def\bbbl@sc@label{generic}%
1821     \def\bbbl@encstring##1##2{%
1822       \ProvideTextCommandDefault##1{##2}%
1823       \bbbl@toglobal##1%
1824       \expandafter\bbbl@toglobal\curname\string?\string##1\endcurname}%

```

```

1825 \let\bb@sctest\in@true
1826 \else
1827 \let\bb@sc@charset\space % <- zapped below
1828 \let\bb@sc@fontenc\space % <- " "
1829 \def\bb@tempa##1=##2\@nil{%
1830 \bb@csarg\edef{sc@zap@space##1 \@empty}{##2 }}%
1831 \bb@vforeach{label=#1}{\bb@tempa##1\@nil}%
1832 \def\bb@tempa##1 ##2{% space -> comma
1833 ##1%
1834 \ifx\@empty##2\else\ifx,##1,\else,\fi\bb@afterfi\bb@tempa##2\fi}%
1835 \edef\bb@sc@fontenc{\expandafter\bb@tempa\bb@sc@fontenc\@empty}%
1836 \edef\bb@sc@label{\expandafter\zap@space\bb@sc@label\@empty}%
1837 \edef\bb@sc@charset{\expandafter\zap@space\bb@sc@charset\@empty}%
1838 \def\bb@encstring##1##2{%
1839 \bb@foreach\bb@sc@fontenc{%
1840 \bb@ifunset{T@###1}%
1841 {}%
1842 {\ProvideTextCommand##1{###1}{##2}%
1843 \bb@tglobal##1%
1844 \expandafter
1845 \bb@tglobal\csname###1\string##1\endcsname}}}%
1846 \def\bb@sctest{%
1847 \bb@xin@{\, \bb@opt@strings,}{, \bb@sc@label, \bb@sc@fontenc,}}%
1848 \fi
1849 \ifx\bb@opt@strings\@nnil % ie, no strings key -> defaults
1850 \else\ifx\bb@opt@strings\relax % ie, strings=encoded
1851 \let\AfterBabelCommands\bb@aftercmds
1852 \let\SetString\bb@setstring
1853 \let\bb@stringdef\bb@encstring
1854 \else % ie, strings=value
1855 \bb@sctest
1856 \ifin@
1857 \let\AfterBabelCommands\bb@aftercmds
1858 \let\SetString\bb@setstring
1859 \let\bb@stringdef\bb@provstring
1860 \fi\fi\fi
1861 \bb@scswitch
1862 \ifx\bb@G\@empty
1863 \def\SetString##1##2{%
1864 \bb@error{Missing group for string \string##1}%
1865 {You must assign strings to some category, typically\%
1866 captions or extras, but you set none}}%
1867 \fi
1868 \ifx\@empty#1%
1869 \bb@usehooks{defaultcommands}{}%
1870 \else
1871 \@expandtwoargs
1872 \bb@usehooks{encodedcommands}{\bb@sc@charset}{\bb@sc@fontenc}}%
1873 \fi}

```

There are two versions of `\bb@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 (`\bb@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 `\bb@forlang` loops `\bb@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 `\bb@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).

```

1874 \def\bb@forlang#1##2{%
1875 \bb@for#1\bb@L{%
1876 \bb@xin@{, #1,}{, \BabelLanguages,}%
1877 \ifin@#2\relax\fi}}
1878 \def\bb@scswitch{%

```

```

1879 \bbl@forlang\bbl@tempa{%
1880   \ifx\bbl@G\@empty\else
1881     \ifx\SetString\@gobbletwo\else
1882       \edef\bbl@GL{\bbl@G\bbl@tempa}%
1883       \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
1884       \ifin@else
1885         \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1886         \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1887       \fi
1888     \fi
1889   \fi}}
1890 \AtEndOfPackage{%
1891   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{#2}}}%
1892   \let\bbl@scswitch\relax}
1893 \@onlypreamble\EndBabelCommands
1894 \def\EndBabelCommands{%
1895   \bbl@usehooks{stopcommands}{}%
1896   \endgroup
1897   \endgroup
1898   \bbl@scafter}
1899 \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.

```

1900 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1901   \bbl@forlang\bbl@tempa{%
1902     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1903     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1904     {\bbl@exp{%
1905       \global\\bbl@add\<\bbl@G\bbl@tempa>{\\bbl@scset\\#1\<\bbl@LC>}}}%
1906     }%
1907     \def\BabelString{#2}%
1908     \bbl@usehooks{stringprocess}{}%
1909     \expandafter\bbl@stringdef
1910     \csname\bbl@LC\expandafter\endcsname\expandafter{\BabelString}}

```

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

```
1911 \def\bbl@scset#1#2{\def#1{#2}}
```

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

```

1912 <<(*Macros local to BabelCommands)>> ≡
1913 \def\SetStringLoop##1##2{%
1914   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1915   \count@\z@
1916   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1917     \advance\count@\@ne
1918     \toks@\expandafter{\bbl@tempa}%
1919     \bbl@exp{%
1920       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1921       \count@=\the\count@\relax}}}%
1922 <</Macros local to BabelCommands>>

```

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

```

1923 \def\bbl@aftercmds#1{%
1924   \toks@\expandafter{\bbl@scafter#1}%
1925   \xdef\bbl@scafter{\the\toks@}

```

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

```

1926 <<(*Macros local to BabelCommands)>> ≡
1927 \newcommand\SetCase[3][]{%
1928   \def\bb@tempa####1####2{%
1929     \ifx####1\@empty\else
1930       \bb@carg\bb@add{extras\CurrentOption}{%
1931         \bb@carg\babel@save{c__text_uppercase_\string####1_tl}%
1932         \bb@carg\def{c__text_uppercase_\string####1_tl}{####2}%
1933         \bb@carg\babel@save{c__text_lowercase_\string####2_tl}%
1934         \bb@carg\def{c__text_lowercase_\string####2_tl}{####1}}%
1935       \expandafter\bb@tempa
1936     \fi}%
1937   \bb@tempa##1\@empty\@empty
1938   \bb@carg\bb@tglobal{extras\CurrentOption}}
1939 <</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.

```

1940 <<(*Macros local to BabelCommands)>> ≡
1941 \newcommand\SetHyphenMap[1]{%
1942   \bb@forlang\bb@tempa{%
1943     \expandafter\bb@stringdef
1944     \csname\bb@tempa @bb@hyphenmap\endcsname{##1}}}%
1945 <</Macros local to BabelCommands>>

```

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

```

1946 \newcommand\BabelLower[2]{% one to one.
1947   \ifnum\lccode#1=#2\else
1948     \babel@savevariable{\lccode#1}%
1949     \lccode#1=#2\relax
1950   \fi}
1951 \newcommand\BabelLowerMM[4]{% many-to-many
1952   \@tempcnta=#1\relax
1953   \@tempcntb=#4\relax
1954   \def\bb@tempa{%
1955     \ifnum\@tempcnta>#2\else
1956       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1957       \advance\@tempcnta#3\relax
1958       \advance\@tempcntb#3\relax
1959       \expandafter\bb@tempa
1960     \fi}%
1961   \bb@tempa}
1962 \newcommand\BabelLowerM0[4]{% many-to-one
1963   \@tempcnta=#1\relax
1964   \def\bb@tempa{%
1965     \ifnum\@tempcnta>#2\else
1966       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1967       \advance\@tempcnta#3
1968       \expandafter\bb@tempa
1969     \fi}%
1970   \bb@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1971 <<(*More package options)>> ≡
1972 \DeclareOption{hyphenmap=off}{\chardef\bb@opt@hyphenmap\z@}
1973 \DeclareOption{hyphenmap=first}{\chardef\bb@opt@hyphenmap\@ne}
1974 \DeclareOption{hyphenmap=select}{\chardef\bb@opt@hyphenmap\tw@}
1975 \DeclareOption{hyphenmap=other}{\chardef\bb@opt@hyphenmap\thr@}
1976 \DeclareOption{hyphenmap=other*}{\chardef\bb@opt@hyphenmap4\relax}
1977 <</More package options>>

```


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

```

1978 \AtEndOfPackage{%
1979   \ifx\bbbl@opt@hyphenmap\@undefined
1980     \bbbl@xin@{,}{\bbbl@language@opts}%
1981     \chardef\bbbl@opt@hyphenmap\ifin@4\else\@ne\fi
1982   \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.

```

1983 \newcommand\setlocalecaption{% TODO. Catch typos.
1984   \@ifstar\bbbl@setcaption@s\bbbl@setcaption@x}
1985 \def\bbbl@setcaption@x#1#2#3{% language caption-name string
1986   \bbbl@trim@def\bbbl@tempa{#2}%
1987   \bbbl@xin@{.template}{\bbbl@tempa}%
1988   \ifin@
1989     \bbbl@ini@captions@template{#3}{#1}%
1990   \else
1991     \edef\bbbl@tempd{%
1992       \expandafter\expandafter\expandafter
1993       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1994     \bbbl@xin@
1995       {\expandafter\string\csname #2name\endcsname}%
1996       {\bbbl@tempd}%
1997     \ifin@ % Renew caption
1998       \bbbl@xin@{\string\bbbl@scset}{\bbbl@tempd}%
1999       \ifin@
2000         \bbbl@exp{%
2001           \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2002           {\\bbbl@scset\<#2name>\<#1#2name>}}%
2003           {}}%
2004         \else % Old way converts to new way
2005           \bbbl@ifunset{#1#2name}%
2006             {\bbbl@exp{%
2007               \\bbbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2008               \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2009               {\def\<#2name>{\<#1#2name>}}%
2010               {}}}%
2011             {}%
2012         \fi
2013       \else
2014         \bbbl@xin@{\string\bbbl@scset}{\bbbl@tempd}% New
2015         \ifin@ % New way
2016           \bbbl@exp{%
2017             \\bbbl@add\<captions#1>{\\bbbl@scset\<#2name>\<#1#2name>}}%
2018             \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2019             {\\bbbl@scset\<#2name>\<#1#2name>}}%
2020             {}}%
2021           \else % Old way, but defined in the new way
2022             \bbbl@exp{%
2023               \\bbbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2024               \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2025               {\def\<#2name>{\<#1#2name>}}%
2026               {}}%
2027             \fi%
2028           \fi
2029           \@namedef{#1#2name}{#3}%
2030           \toks@\expandafter{\bbbl@captionslist}%
2031           \bbbl@exp{\\in@{\<#2name>}{\the\toks@}}%
2032           \ifin@\\else
2033             \bbbl@exp{\\bbbl@add\\bbbl@captionslist{\<#2name>}}%
2034             \bbbl@toggle\bbbl@captionslist
2035           \fi

```

```

2036 \fi}
2037 % \def\bb1@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.

```

2038 \bb1@trace{Macros related to glyphs}
2039 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2040   \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2041   \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.

```

2042 \def\save@s@f@q#1{\leavevmode
2043   \begingroup
2044   \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2045   \endgroup}

```

4.12 Making glyphs available

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

4.12.1 Quotation marks

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

```

2046 \ProvideTextCommand{\quotedblbase}{OT1}{%
2047   \save@s@f@q{\set@low@box{\textquotedblright\}}%
2048   \box\z@\kern-.04em\bb1@allowhyphens}}

```

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

```

2049 \ProvideTextCommandDefault{\quotedblbase}{%
2050   \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2051 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2052   \save@s@f@q{\set@low@box{\textquoteright\}}%
2053   \box\z@\kern-.04em\bb1@allowhyphens}}

```

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

```

2054 \ProvideTextCommandDefault{\quotesinglbase}{%
2055   \UseTextSymbol{OT1}{\quotesinglbase}}

```

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

```

2056 \ProvideTextCommand{\guillemetleft}{OT1}{%
2057   \ifmmode
2058     \ll
2059   \else
2060     \save@s@f@q{\nobreak
2061       \raise.2ex\hbox{\scriptscriptstyle\ll}\bb1@allowhyphens}%
2062     \fi}
2063 \ProvideTextCommand{\guillemetright}{OT1}{%
2064   \ifmmode
2065     \gg
2066   \else
2067     \save@s@f@q{\nobreak
2068       \raise.2ex\hbox{\scriptscriptstyle\gg}\bb1@allowhyphens}%
2069     \fi}
2070 \ProvideTextCommand{\guillemotleft}{OT1}{%

```

```

2071 \ifmmode
2072   \ll
2073 \else
2074   \save@sf@q{\nobreak
2075     \raise.2ex\hbox{\scriptscriptstyle\ll}\bbl@allowhyphens}%
2076 \fi}
2077 \ProvideTextCommand{\guillemotright}{OT1}{%
2078 \ifmmode
2079   \gg
2080 \else
2081   \save@sf@q{\nobreak
2082     \raise.2ex\hbox{\scriptscriptstyle\gg}\bbl@allowhyphens}%
2083 \fi}

```

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

```

2084 \ProvideTextCommandDefault{\guillemetleft}{%
2085 \UseTextSymbol{OT1}{\guillemetleft}}
2086 \ProvideTextCommandDefault{\guillemetright}{%
2087 \UseTextSymbol{OT1}{\guillemetright}}
2088 \ProvideTextCommandDefault{\guillemotleft}{%
2089 \UseTextSymbol{OT1}{\guillemotleft}}
2090 \ProvideTextCommandDefault{\guillemotright}{%
2091 \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

\guilsinglright 2092 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2093 \ifmmode
2094   <%
2095 \else
2096   \save@sf@q{\nobreak
2097     \raise.2ex\hbox{\scriptscriptstyle<}\bbl@allowhyphens}%
2098 \fi}
2099 \ProvideTextCommand{\guilsinglright}{OT1}{%
2100 \ifmmode
2101   >%
2102 \else
2103   \save@sf@q{\nobreak
2104     \raise.2ex\hbox{\scriptscriptstyle>}\bbl@allowhyphens}%
2105 \fi}

```

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

```

2106 \ProvideTextCommandDefault{\guilsinglleft}{%
2107 \UseTextSymbol{OT1}{\guilsinglleft}}
2108 \ProvideTextCommandDefault{\guilsinglright}{%
2109 \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.

```

2110 \DeclareTextCommand{\ij}{OT1}{%
2111 i\kern-0.02em\bbl@allowhyphens j}
2112 \DeclareTextCommand{\IJ}{OT1}{%
2113 I\kern-0.02em\bbl@allowhyphens J}
2114 \DeclareTextCommand{\ij}{T1}{\char188}
2115 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2116 \ProvideTextCommandDefault{\ij}{%
2117 \UseTextSymbol{OT1}{\ij}}
2118 \ProvideTextCommandDefault{\IJ}{%
2119 \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).

```

2120 \def\crrtic@{\hrule height0.1ex width0.3em}
2121 \def\crrtic@{\hrule height0.1ex width0.33em}
2122 \def\ddj@{%
2123   \setbox0\hbox{d}\dimen@=\ht0
2124   \advance\dimen@lex
2125   \dimen@.45\dimen@
2126   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2127   \advance\dimen@ii.5ex
2128   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}
2129 \def\DDJ@{%
2130   \setbox0\hbox{D}\dimen@=.55\ht0
2131   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2132   \advance\dimen@ii.15ex % correction for the dash position
2133   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2134   \dimen\thr@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2135   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}
2136 %
2137 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2138 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2139 \ProvideTextCommandDefault{\dj}{%
2140   \UseTextSymbol{OT1}{\dj}}
2141 \ProvideTextCommandDefault{\DJ}{%
2142   \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.

```

2143 \DeclareTextCommand{\SS}{OT1}{SS}
2144 \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
2145 \ProvideTextCommandDefault{\glq}{%
2146   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

```

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

```

2147 \ProvideTextCommand{\grq}{T1}{%
2148   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2149 \ProvideTextCommand{\grq}{TU}{%
2150   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2151 \ProvideTextCommand{\grq}{OT1}{%
2152   \save@sf@q{\kern-.0125em
2153     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2154     \kern.07em\relax}}
2155 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}

```

`\glqq` The ‘german’ double quotes.

```

\grqq
2156 \ProvideTextCommandDefault{\glqq}{%
2157   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

2158 \ProvideTextCommand{\grqq}{T1}{%
2159   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}

```

```

2160 \ProvideTextCommand{\grqq}{TU}{%
2161   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2162 \ProvideTextCommand{\grqq}{OT1}{%
2163   \save@sf@q{\kern-.07em
2164     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2165     \kern.07em\relax}}
2166 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}

```

`\flq` The ‘french’ single guillemets.

```

\frq 2167 \ProvideTextCommandDefault{\flq}{%
2168   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2169 \ProvideTextCommandDefault{\frq}{%
2170   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

`\flqq` The ‘french’ double guillemets.

```

\frqq 2171 \ProvideTextCommandDefault{\flqq}{%
2172   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2173 \ProvideTextCommandDefault{\frqq}{%
2174   \textormath{\guillemetright}{\mbox{\guillemetright}}}

```

4.12.4 Umlauts and tremas

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

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

```

2175 \def\umlauthigh{%
2176   \def\bbl@umlauta##1{\leavevmode\bgroup%
2177     \accent\csname\fontencoding dqpos\endcsname
2178     ##1\bbl@allowhyphens\egroup}%
2179   \let\bbl@umlaute\bbl@umlauta}
2180 \def\umlautlow{%
2181   \def\bbl@umlauta{\protect\lower@umlaut}}
2182 \def\umlautelaw{%
2183   \def\bbl@umlaute{\protect\lower@umlaut}}
2184 \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.

```

2185 \expandafter\ifx\csname U@D\endcsname\relax
2186   \csname newdimen\endcsname\U@D
2187 \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.

```

2188 \def\lower@umlaut#1{%
2189   \leavevmode\bgroup
2190   \U@D lex%
2191   {\setbox\z@\hbox{%
2192     \char\csname\fontencoding dqpos\endcsname}%
2193     \dimen@ -.45ex\advance\dimen@\ht\z@
2194     \ifdim lex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2195   \accent\csname\fontencoding dqpos\endcsname
2196   \fontdimen5\font\U@D #1%
2197   \egroup}

```

For all vowels we declare `\` to be a composite command which uses `\bbl@umlauta` or `\bbl@umlaute` to position the umlaut character. We need to be sure that these definitions override the ones that are provided when the package `fontenc` with option `OT1` is used. Therefore these declarations are postponed until the beginning of the document. Note these definitions only apply to some languages, but `babel` sets them for *all* languages – you may want to redefine `\bbl@umlauta` and/or `\bbl@umlaute` for a language in the corresponding `ldf` (using the `babel` switching mechanism, of course).

```

2198 \AtBeginDocument{%
2199   \DeclareTextCompositeCommand{"}{OT1}{a}{\bbl@umlauta{a}}%
2200   \DeclareTextCompositeCommand{"}{OT1}{e}{\bbl@umlaute{e}}%
2201   \DeclareTextCompositeCommand{"}{OT1}{i}{\bbl@umlaute{i}}%
2202   \DeclareTextCompositeCommand{"}{OT1}{\i}{\bbl@umlaute{i}}%
2203   \DeclareTextCompositeCommand{"}{OT1}{o}{\bbl@umlauta{o}}%
2204   \DeclareTextCompositeCommand{"}{OT1}{u}{\bbl@umlauta{u}}%
2205   \DeclareTextCompositeCommand{"}{OT1}{A}{\bbl@umlauta{A}}%
2206   \DeclareTextCompositeCommand{"}{OT1}{E}{\bbl@umlaute{E}}%
2207   \DeclareTextCompositeCommand{"}{OT1}{I}{\bbl@umlaute{I}}%
2208   \DeclareTextCompositeCommand{"}{OT1}{O}{\bbl@umlauta{O}}%
2209   \DeclareTextCompositeCommand{"}{OT1}{U}{\bbl@umlauta{U}}

```

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

```

2210 \ifx\l@english\undefined
2211   \chardef\l@english\z@
2212 \fi
2213 % The following is used to cancel rules in ini files (see Amharic).
2214 \ifx\l@unhyphenated\undefined
2215   \newlanguage\l@unhyphenated
2216 \fi

```

4.13 Layout

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

```

2217 \bbl@trace{Bidi layout}
2218 \providecommand\IfBabelLayout[3]{#3}%
2219 {-core}
2220 \newcommand\BabelPatchSection[1]{%
2221   \@ifundefined{#1}{}%
2222   \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2223   \@namedef{#1}{%
2224     \ifstar{\bbl@presec@s{#1}}%
2225     {\@dblarg{\bbl@presec@x{#1}}}}%
2226 \def\bbl@presec@x#1[#2]#3{%
2227   \bbl@exp{%
2228     \\select@language@x{\bbl@main@language}%
2229     \\bbl@cs{sspre@#1}%
2230     \\bbl@cs{ss@#1}%
2231     [\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
2232     {\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
2233     \\select@language@x{\languagename}}%
2234 \def\bbl@presec@s#1#2{%
2235   \bbl@exp{%
2236     \\select@language@x{\bbl@main@language}%
2237     \\bbl@cs{sspre@#1}%
2238     \\bbl@cs{ss@#1}*%
2239     {\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
2240     \\select@language@x{\languagename}}%
2241 \IfBabelLayout{sectioning}%
2242 {\BabelPatchSection{part}}%
2243 \BabelPatchSection{chapter}%
2244 \BabelPatchSection{section}%
2245 \BabelPatchSection{subsection}%
2246 \BabelPatchSection{subsubsection}%

```

```

2247 \BabelPatchSection{paragraph}%
2248 \BabelPatchSection{subparagraph}%
2249 \def\babel@toc#1{%
2250 \select@language@x{\bbl@main@language}}{}
2251 \IfBabelLayout{captions}%
2252 {\BabelPatchSection{caption}}{}
2253 <+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:

```

2254 \bbl@trace{Input engine specific macros}
2255 \ifcase\bbl@engine
2256 \input txtbabel.def
2257 \or
2258 \input luababel.def
2259 \or
2260 \input xebabel.def
2261 \fi
2262 \providecommand\babelfont{%
2263 \bbl@error
2264 {This macro is available only in LuaLaTeX and XeLaTeX.}%
2265 {Consider switching to these engines.}}
2266 \providecommand\babelprehyphenation{%
2267 \bbl@error
2268 {This macro is available only in LuaLaTeX.}%
2269 {Consider switching to that engine.}}
2270 \ifx\babelposthyphenation\undefined
2271 \let\babelposthyphenation\babelprehyphenation
2272 \let\babelpatterns\babelprehyphenation
2273 \let\babelcharproperty\babelprehyphenation
2274 \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.

```

2275 </package | core>
2276 <*package>
2277 \bbl@trace{Creating languages and reading ini files}
2278 \let\bbl@extend@ini@gobble
2279 \newcommand\babelprovide[2][{}]{%
2280 \let\bbl@savelangname\languagename
2281 \edef\bbl@savelocaleid{\the\localeid}%
2282 % Set name and locale id
2283 \edef\languagename{#2}%
2284 \bbl@id@assign
2285 % Initialize keys
2286 \bbl@vforeach{captions,date,import,main,script,language,%
2287 hyphenrules,linebreaking,justification,mapfont,maparabic,%
2288 mapdigits,intraspace,intrapenalty,onchar,transforms,alph,%
2289 Alph,labels,labels*,calendar,date,casing}%
2290 {\bbl@csarg\let{KVP@##1}\@nnil}%
2291 \global\let\bbl@release@transforms@empty
2292 \global\let\bbl@release@casing@empty
2293 \let\bbl@calendars@empty
2294 \global\let\bbl@inidata@empty
2295 \global\let\bbl@extend@ini@gobble
2296 \global\let\bbl@included@inis@empty

```

```

2297 \gdef\bb@key@list{;}%
2298 \bb@forkv{#1}{%
2299   \in{/}{##1}% With /, (re)sets a value in the ini
2300   \ifin@
2301     \global\let\bb@extend@ini\bb@extend@ini@aux
2302     \bb@renewinikey##1\@{##2}%
2303   \else
2304     \bb@csarg\ifx{KVP@##1}\@nnil\else
2305       \bb@error
2306         {Unknown key '##1' in \string\babelprovide}%
2307         {See the manual for valid keys}%
2308     \fi
2309     \bb@csarg\def{KVP@##1}{##2}%
2310   \fi}%
2311 \chardef\bb@howloaded=% 0:none; 1:ldf without ini; 2:ini
2312 \bb@ifunset{date#2}\z@{\bb@ifunset{\bb@llevel#2}\ne\tw@}%
2313 % == init ==
2314 \ifx\bb@screset\@undefined
2315   \bb@ldfinit
2316 \fi
2317 % == date (as option) ==
2318 % \ifx\bb@KVP@date\@nnil\else
2319 % \fi
2320 % ==
2321 \let\bb@lbfkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2322 \ifcase\bb@howloaded
2323   \let\bb@lbfkflag\@empty % new
2324 \else
2325   \ifx\bb@KVP@hyphenrules\@nnil\else
2326     \let\bb@lbfkflag\@empty
2327   \fi
2328   \ifx\bb@KVP@import\@nnil\else
2329     \let\bb@lbfkflag\@empty
2330   \fi
2331 \fi
2332 % == import, captions ==
2333 \ifx\bb@KVP@import\@nnil\else
2334   \bb@exp{\bb@ifblank{\bb@KVP@import}}%
2335   {\ifx\bb@initload\relax
2336     \begingroup
2337       \def\BabelBeforeIni##1##2{\gdef\bb@KVP@import{##1}\endinput}%
2338       \bb@input@texini{##2}%
2339     \endgroup
2340   \else
2341     \xdef\bb@KVP@import{\bb@initload}%
2342   \fi}%
2343 {}%
2344 \let\bb@KVP@date\@empty
2345 \fi
2346 \let\bb@KVP@captions@\bb@KVP@captions % TODO. A dirty hack
2347 \ifx\bb@KVP@captions\@nnil
2348   \let\bb@KVP@captions\bb@KVP@import
2349 \fi
2350 % ==
2351 \ifx\bb@KVP@transforms\@nnil\else
2352   \bb@replace\bb@KVP@transforms{ }{,}%
2353 \fi
2354 % == Load ini ==
2355 \ifcase\bb@howloaded
2356   \bb@provide@new{##2}%
2357 \else
2358   \bb@ifblank{#1}%
2359   {}% With \bb@load@basic below

```



```

2360     {\bbl@provide@renew{#2}}%
2361 \fi
2362 % == include == TODO
2363 % \ifx\bbl@included@inis\@empty\else
2364 %   \bbl@replace\bbl@included@inis{ },}%
2365 %   \bbl@foreach\bbl@included@inis{%
2366 %     \openin\bbl@readstream=babel-##1.ini
2367 %     \bbl@extend@ini{#2}}%
2368 %   \closein\bbl@readstream
2369 % \fi
2370 % Post tasks
2371 % -----
2372 % == subsequent calls after the first provide for a locale ==
2373 \ifx\bbl@inidata\@empty\else
2374   \bbl@extend@ini{#2}%
2375 \fi
2376 % == ensure captions ==
2377 \ifx\bbl@KVP@captions\@nnil\else
2378   \bbl@ifunset{bbl@extracaps@#2}%
2379     {\bbl@exp{\\babelensure[exclude=\\today]{#2}}}%
2380     {\bbl@exp{\\babelensure[exclude=\\today,
2381               include=\[bbl@extracaps@#2]]{#2}}}%
2382   \bbl@ifunset{bbl@ensure@language}%
2383     {\bbl@exp{%
2384       \\DeclareRobustCommand\<bbl@ensure@language>[1]{%
2385         \\foreignlanguage{language}%
2386         {###1}}}%
2387     }%
2388   \bbl@exp{%
2389     \\bbl@tglobal\<bbl@ensure@language>%
2390     \\bbl@tglobal\<bbl@ensure@language\space>%
2391 \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.

```

2392 \bbl@load@basic{#2}%
2393 % == script, language ==
2394 % Override the values from ini or defines them
2395 \ifx\bbl@KVP@script\@nnil\else
2396   \bbl@csarg\edef{sname#2}{\bbl@KVP@script}%
2397 \fi
2398 \ifx\bbl@KVP@language\@nnil\else
2399   \bbl@csarg\edef{lname#2}{\bbl@KVP@language}%
2400 \fi
2401 \ifcase\bbl@engine\or
2402   \bbl@ifunset{bbl@chrng@language}{}%
2403   {\directlua{
2404     Babel.set_chranges_b('\bbl@cl{sbcpr}', '\bbl@cl{chrng}') }}%
2405 \fi
2406 % == onchar ==
2407 \ifx\bbl@KVP@onchar\@nnil\else
2408   \bbl@lua\hyphenate
2409   \bbl@exp{%
2410     \\AddToHook{env/document/before}{\\select@language{#2}}}%
2411   \directlua{
2412     if Babel.locale_mapped == nil then
2413       Babel.locale_mapped = true
2414       Babel.linebreaking.add_before(Babel.locale_map, 1)
2415       Babel.loc_to_scr = {}
2416       Babel.chr_to_loc = Babel.chr_to_loc or {}
2417     end
2418     Babel.locale_props[\the\localeid].letters = false

```

```

2419 }%
2420 \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
2421 \ifin@
2422   \directlua{
2423     Babel.locale_props[\the\localeid].letters = true
2424   }%
2425 \fi
2426 \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2427 \ifin@
2428   \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
2429     \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
2430   \fi
2431   \bbl@exp{\bbl@add\bbl@starthyphens
2432     {\bbl@patterns@lua{\languagename}}}%
2433   % TODO - error/warning if no script
2434   \directlua{
2435     if Babel.script_blocks['\bbl@cl{sbc}'] then
2436       Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbl@cl{sbc}']
2437       Babel.locale_props[\the\localeid].lg = \the\@nameuse{\languagename}\space
2438     end
2439   }%
2440 \fi
2441 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2442 \ifin@
2443   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}}%
2444   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}}%
2445   \directlua{
2446     if Babel.script_blocks['\bbl@cl{sbc}'] then
2447       Babel.loc_to_scr[\the\localeid] =
2448         Babel.script_blocks['\bbl@cl{sbc}']
2449     end}%
2450   \ifx\bbl@mapselect\@undefined % TODO. almost the same as mapfont
2451     \AtBeginDocument{%
2452       \bbl@patchfont{\bbl@mapselect}%
2453       {\selectfont}}%
2454     \def\bbl@mapselect{%
2455       \let\bbl@mapselect\relax
2456       \edef\bbl@prefontid{\fontid\font}}%
2457     \def\bbl@mapdir##1{%
2458       {\def\languagename{##1}%
2459       \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2460       \bbl@switchfont
2461       \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2462         \directlua{
2463           Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
2464             [/\bbl@prefontid'] = \fontid\font\space}%
2465         \fi}}%
2466     \fi
2467     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languagename}}}%
2468   \fi
2469   % TODO - catch non-valid values
2470 \fi
2471 % == mapfont ==
2472 % For bidi texts, to switch the font based on direction
2473 \ifx\bbl@KVP@mapfont\@nnil\else
2474   \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}}%
2475   {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\%
2476     mapfont. Use 'direction'.%
2477     {See the manual for details.}}}%
2478   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}}%
2479   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}}%
2480 \ifx\bbl@mapselect\@undefined % TODO. See onchar.
2481   \AtBeginDocument{%

```

```

2482     \bbl@patchfont{{\bbl@mapselect}}%
2483     {\selectfont}}%
2484     \def\bbl@mapselect{%
2485         \let\bbl@mapselect\relax
2486         \edef\bbl@prefontid{\fontid\font}}%
2487     \def\bbl@mapdir##1{%
2488         {\def\languagename{##1}%
2489         \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2490         \bbl@switchfont
2491         \directlua{Babel.fontmap
2492         [\the\csname bbl@wdir@##1\endcsname]%
2493         [\bbl@prefontid]=\fontid\font}}}%
2494     \fi
2495     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languagename}}}%
2496     \fi
2497     % == Line breaking: intraspace, intrapenalty ==
2498     % For CJK, East Asian, Southeast Asian, if interspace in ini
2499     \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2500         \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2501     \fi
2502     \bbl@provide@intraspace
2503     % == Line breaking: CJK quotes == TODO -> @extras
2504     \ifcase\bbl@engine\or
2505         \bbl@xin@{/c}{/\bbl@cl{\lnbrk}}%
2506     \ifin@
2507         \bbl@ifunset{bbl@quote@\languagename}{}%
2508         {\directlua{
2509             Babel.locale_props[\the\localeid].cjk_quotes = {}
2510             local cs = 'op'
2511             for c in string.utfvalues(
2512                 [[\csname bbl@quote@\languagename\endcsname]]) do
2513                 if Babel.cjk_characters[c].c == 'qu' then
2514                     Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2515                 end
2516                 cs = ( cs == 'op' ) and 'cl' or 'op'
2517             end
2518         }}%
2519     \fi
2520     \fi
2521     % == Line breaking: justification ==
2522     \ifx\bbl@KVP@justification\@nnil\else
2523         \let\bbl@KVP@linebreaking\bbl@KVP@justification
2524     \fi
2525     \ifx\bbl@KVP@linebreaking\@nnil\else
2526         \bbl@xin@{,\bbl@KVP@linebreaking,}%
2527         {,elongated,kashida,cjk,padding,unhyphenated,}%
2528     \ifin@
2529         \bbl@csarg\xdef
2530             {\lnbrk@\languagename}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2531     \fi
2532     \fi
2533     \bbl@xin@{/e}{/\bbl@cl{\lnbrk}}%
2534     \ifin@\else\bbl@xin@{/k}{/\bbl@cl{\lnbrk}}\fi
2535     \ifin@\bbl@arabicjust\fi
2536     \bbl@xin@{/p}{/\bbl@cl{\lnbrk}}%
2537     \ifin@\AtBeginDocument{\@nameuse{bbl@tibetanjust}}\fi
2538     % == Line breaking: hyphenate.other.(locale|script) ==
2539     \ifx\bbl@lbfkflag\@empty
2540         \bbl@ifunset{bbl@hyotl@\languagename}{}%
2541         {\bbl@csarg\bbl@replace{hyotl@\languagename}{ }{,}}%
2542         \bbl@startcommands*\languagename}{}%
2543         \bbl@csarg\bbl@foreach{hyotl@\languagename}{%
2544             \ifcase\bbl@engine

```

```

2545         \ifnum##1<257
2546         \SetHyphenMap{\BabelLower{##1}{##1}}%
2547         \fi
2548         \else
2549         \SetHyphenMap{\BabelLower{##1}{##1}}%
2550         \fi}%
2551     \bbl@endcommands}%
2552 \bbl@ifunset{\bbl@hyots@\languagename}{}%
2553 {\bbl@csarg\bbl@replace{\hyots@\languagename}{ }{,}}%
2554 \bbl@csarg\bbl@foreach{\hyots@\languagename}{%
2555     \ifcase\bbl@engine
2556     \ifnum##1<257
2557     \global\lccode##1=##1\relax
2558     \fi
2559     \else
2560     \global\lccode##1=##1\relax
2561     \fi}}%
2562 \fi
2563 % == Counters: maparabic ==
2564 % Native digits, if provided in ini (TeX level, xe and lua)
2565 \ifcase\bbl@engine\else
2566     \bbl@ifunset{\bbl@dgnat@\languagename}{}%
2567     {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
2568     \expandafter\expandafter\expandafter
2569     \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname
2570     \ifx\bbl@KVP@maparabic\@nnil\else
2571     \ifx\bbl@latinarabic\@undefined
2572     \expandafter\let\expandafter\@arabic
2573     \csname bbl@counter@\languagename\endcsname
2574     \else % ie, if layout=counters, which redefines \@arabic
2575     \expandafter\let\expandafter\bbl@latinarabic
2576     \csname bbl@counter@\languagename\endcsname
2577     \fi
2578     \fi
2579     \fi}%
2580 \fi
2581 % == Counters: mapdigits ==
2582 % > luababel.def
2583 % == Counters: alph, Alph ==
2584 \ifx\bbl@KVP@alph\@nnil\else
2585     \bbl@exp{%
2586     \\bbl@add\<bbl@preextras@\languagename>{%
2587     \\babel@save\\@alph
2588     \let\\@alph\<bbl@cntr@\bbl@KVP@alph @\languagename>}}%
2589 \fi
2590 \ifx\bbl@KVP@Alph\@nnil\else
2591     \bbl@exp{%
2592     \\bbl@add\<bbl@preextras@\languagename>{%
2593     \\babel@save\\@Alph
2594     \let\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\languagename>}}%
2595 \fi
2596 % == Casing ==
2597 \bbl@release@casing
2598 \ifx\bbl@KVP@casing\@nnil\else
2599     \bbl@csarg\xdef{casing@\languagename}%
2600     {\@nameuse{\bbl@casing@\languagename}\bbl@maybextx\bbl@KVP@casing}%
2601 \fi
2602 % == Calendars ==
2603 \ifx\bbl@KVP@calendar\@nnil
2604     \edef\bbl@KVP@calendar{\bbl@cl{calpr}}%
2605 \fi
2606 \def\bbl@tempe##1 ##2\@{@% Get first calendar
2607     \def\bbl@tempa{##1}}%

```

```

2608 \bbl@exp{\bbl@tempe\bbl@KVP@calendar\space\\@}%
2609 \def\bbl@tempe##1.##2.##3\@{
2610 \def\bbl@tempc{##1}%
2611 \def\bbl@tempb{##2}}%
2612 \expandafter\bbl@tempe\bbl@tempa. \@
2613 \bbl@csarg\edef{calpr@\language}%
2614 \ifx\bbl@tempc\@empty\else
2615 calendar=\bbl@tempc
2616 \fi
2617 \ifx\bbl@tempb\@empty\else
2618 ,variant=\bbl@tempb
2619 \fi}%
2620 % == engine specific extensions ==
2621 % Defined in XXXbabel.def
2622 \bbl@provide@extra{#2}%
2623 % == require.babel in ini ==
2624 % To load or reload the babel-*.tex, if require.babel in ini
2625 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
2626 \bbl@ifunset{\bbl@rqtex@\language}{
2627 {\expandafter\ifx\csname\bbl@rqtex@\language\endcsname\@empty\else
2628 \let\BabelBeforeIni@gobbletwo
2629 \chardef\atcatcode=\catcode\@
2630 \catcode\@=11\relax
2631 \def\CurrentOption{#2}%
2632 \bbl@input@texini{\bbl@cs{rqtex@\language}}%
2633 \catcode\@=\atcatcode
2634 \let\atcatcode\relax
2635 \global\bbl@csarg\let{rqtex@\language}\relax
2636 \fi}%
2637 \bbl@foreach\bbl@calendars{
2638 \bbl@ifunset{\bbl@ca##1}{
2639 \chardef\atcatcode=\catcode\@
2640 \catcode\@=11\relax
2641 \InputIfFileExists{babel-ca-##1.tex}{
2642 \catcode\@=\atcatcode
2643 \let\atcatcode\relax}%
2644 }}%
2645 \fi
2646 % == frenchspacing ==
2647 \ifcase\bbl@howloaded\in@true\else\in@false\fi
2648 \ifin@else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2649 \ifin@
2650 \bbl@extras@wrap{\bbl@pre@fs}%
2651 {\bbl@pre@fs}%
2652 {\bbl@post@fs}%
2653 \fi
2654 % == transforms ==
2655 % > luababel.def
2656 % == main ==
2657 \ifx\bbl@KVP@main\@nnil % Restore only if not 'main'
2658 \let\language\bbl@savelangname
2659 \chardef\localeid\bbl@savelocaleid\relax
2660 \fi
2661 % == hyphenrules (apply if current) ==
2662 \ifx\bbl@KVP@hyphenrules\@nnil\else
2663 \ifnum\bbl@savelocaleid=\localeid
2664 \language\@nameuse{l@\language}%
2665 \fi
2666 \fi}

```

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

```
2667 \def\bbl@provide@new#1{%
```

```

2668 \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2669 \@namedef{extras#1}{}%
2670 \@namedef{noextras#1}{}%
2671 \bbl@startcommands*{#1}{captions}%
2672 \ifx\bbl@KVP@captions\@nnil % and also if import, implicit
2673 \def\bbl@tempb##1{% elt for \bbl@captionslist
2674 \ifx##1\@empty\else
2675 \bbl@exp{%
2676 \\SetString\\##1{%
2677 \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2678 \expandafter\bbl@tempb
2679 \fi}%
2680 \expandafter\bbl@tempb\bbl@captionslist\@empty
2681 \else
2682 \ifx\bbl@initoload\relax
2683 \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
2684 \else
2685 \bbl@read@ini{\bbl@initoload}2% % Same
2686 \fi
2687 \fi
2688 \StartBabelCommands*{#1}{date}%
2689 \ifx\bbl@KVP@date\@nnil
2690 \bbl@exp{%
2691 \\SetString\\today{\bbl@nocaption{today}{#1today}}}%
2692 \else
2693 \bbl@savetoday
2694 \bbl@savedate
2695 \fi
2696 \bbl@endcommands
2697 \bbl@load@basic{#1}%
2698 % == hyphenmins == (only if new)
2699 \bbl@exp{%
2700 \gdef<#1hyphenmins>{%
2701 {\bbl@ifunset{\bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
2702 {\bbl@ifunset{\bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
2703 % == hyphenrules (also in renew) ==
2704 \bbl@provide@hyphens{#1}%
2705 \ifx\bbl@KVP@main\@nnil\else
2706 \expandafter\main@language\expandafter{#1}%
2707 \fi}
2708 %
2709 \def\bbl@provide@renew#1{%
2710 \ifx\bbl@KVP@captions\@nnil\else
2711 \StartBabelCommands*{#1}{captions}%
2712 \bbl@read@ini{\bbl@KVP@captions}2% % Here all letters cat = 11
2713 \EndBabelCommands
2714 \fi
2715 \ifx\bbl@KVP@date\@nnil\else
2716 \StartBabelCommands*{#1}{date}%
2717 \bbl@savetoday
2718 \bbl@savedate
2719 \EndBabelCommands
2720 \fi
2721 % == hyphenrules (also in new) ==
2722 \ifx\bbl@lbkflag\@empty
2723 \bbl@provide@hyphens{#1}%
2724 \fi}
2725 \def\bbl@load@basic#1{%
2726 \ifcase\bbl@howloaded\or\or

```

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.)

```

2727 \ifcase\csname bbl@llevel@\languagename\endcsname
2728 \bbl@csarg\let\lname@\languagename\relax
2729 \fi
2730 \fi
2731 \bbl@ifunset{bbl@lname@#1}%
2732 {\def\BabelBeforeIni##1##2{%
2733 \begingroup
2734 \let\bbl@ini@captions@aux\@gobbletwo
2735 \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6}%
2736 \bbl@read@ini{##1}1%
2737 \ifx\bbl@initoload\relax\endinput\fi
2738 \endgroup}%
2739 \begingroup % boxed, to avoid extra spaces:
2740 \ifx\bbl@initoload\relax
2741 \bbl@input@texini{##1}%
2742 \else
2743 \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
2744 \fi
2745 \endgroup}%
2746 {}

```

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`.

```

2747 \def\bbl@provide@hyphens#1{%
2748 \@tempcnta\m@ne % a flag
2749 \ifx\bbl@KVP@hyphenrules\@nnil\else
2750 \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2751 \bbl@foreach\bbl@KVP@hyphenrules{%
2752 \ifnum\@tempcnta=\m@ne % if not yet found
2753 \bbl@ifsamestring{##1}{+}%
2754 {\bbl@carg\addlanguage{l@##1}}%
2755 }%
2756 \bbl@ifunset{l@##1}% After a possible +
2757 {}%
2758 {\@tempcnta\@nameuse{l@##1}}%
2759 \fi}%
2760 \ifnum\@tempcnta=\m@ne
2761 \bbl@warning{%
2762 Requested 'hyphenrules' for '\languagename' not found:\\%
2763 \bbl@KVP@hyphenrules.\\%
2764 Using the default value. Reported}%
2765 \fi
2766 \fi
2767 \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found
2768 \ifx\bbl@KVP@captions@\@nnil % TODO. Hackish. See above.
2769 \bbl@ifunset{bbl@hyphr@#1}{}% use value in ini, if exists
2770 {\bbl@exp{\bbl@ifblank{\bbl@cs{hyphr@#1}}}%
2771 }%
2772 {\bbl@ifunset{l@\bbl@cl{hyphr}}}%
2773 }% if hyphenrules found:
2774 {\@tempcnta\@nameuse{l@\bbl@cl{hyphr}}}%
2775 \fi
2776 \fi
2777 \bbl@ifunset{l@#1}%
2778 {\ifnum\@tempcnta=\m@ne
2779 \bbl@carg\adddialect{l@#1}\language
2780 \else
2781 \bbl@carg\adddialect{l@#1}\@tempcnta
2782 \fi}%
2783 {\ifnum\@tempcnta=\m@ne\else
2784 \global\bbl@carg\chardef{l@#1}\@tempcnta
2785 \fi}}

```

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

```

2786 \def\bbl@input@texini#1{%
2787   \bbl@bsphack
2788   \bbl@exp{%
2789     \catcode`\\%=14 \catcode`\\%=0
2790     \catcode`\\{=1 \catcode`\\}=2
2791     \lowercase{\InputIfFileExists{babel-#1.tex}{}}}%
2792     \catcode`\\%=the\catcode`%\relax
2793     \catcode`\\%=the\catcode`%\relax
2794     \catcode`\\{=the\catcode`%\relax
2795     \catcode`\\}=the\catcode`%\relax}%
2796   \bbl@esphack}

```

The following macros read and store ini files (but don't process them). For each line, there are 3 possible actions: ignore if starts with ;, switch section if starts with [, and store otherwise. There are used in the first step of \bbl@read@ini.

```

2797 \def\bbl@iniline#1\bbl@iniline{%
2798   \@ifnextchar[\bbl@inisect{\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
2799 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2800 \def\bbl@iniskip#1\@@{%      if starts with ;
2801 \def\bbl@inistore#1=#2\@@{%  full (default)
2802   \bbl@trim@def\bbl@tempa{#1}%
2803   \bbl@trim\toks@{#2}%
2804   \bbl@xin@{;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2805   \ifin@else
2806     \bbl@xin@{,identification/include.}%
2807     {,\bbl@section/\bbl@tempa}%
2808     \ifin@\xdef\bbl@included@inis{\the\toks@}\fi
2809     \bbl@exp{%
2810       \\g@addto@macro\\bbl@inidata{%
2811         \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2812     \fi}
2813 \def\bbl@inistore@min#1=#2\@@{%  minimal (maybe set in \bbl@read@ini)
2814   \bbl@trim@def\bbl@tempa{#1}%
2815   \bbl@trim\toks@{#2}%
2816   \bbl@xin@{.identification.}{.\bbl@section.}%
2817   \ifin@
2818     \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2819       \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2820   \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.

```

2821 \def\bbl@loop@ini{%
2822   \loop
2823     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2824     \endlinechar\m@ne
2825     \read\bbl@readstream to \bbl@line
2826     \endlinechar`^^M
2827     \ifx\bbl@line@empty\else
2828       \expandafter\bbl@iniline\bbl@line\bbl@iniline
2829     \fi
2830     \repeat}
2831 \ifx\bbl@readstream@undefined
2832   \csname newread\endcsname\bbl@readstream
2833 \fi
2834 \def\bbl@read@ini#1#2{%
2835   \global\let\bbl@extend@ini@gobble
2836   \openin\bbl@readstream=babel-#1.ini
2837   \ifeof\bbl@readstream
2838     \bbl@error

```



```

2839     {There is no ini file for the requested language\\%
2840     (#1: \languagename). Perhaps you misspelled it or your\\%
2841     installation is not complete.}%
2842     {Fix the name or reinstall babel.}%
2843 \else
2844     % == Store ini data in \bbl@inidata ==
2845     \catcode`\[=12 \catcode`\]=12 \catcode`\&=12 \catcode`\&=12
2846     \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2847     \bbl@info{Importing
2848         \ifcase#2font and identification \or basic \fi
2849         data for \languagename\\%
2850         from babel-#1.ini. Reported}%
2851     \ifnum#2=\z@
2852         \global\let\bbl@inidata@empty
2853         \let\bbl@inistore\bbl@inistore@min    % Remember it's local
2854     \fi
2855     \def\bbl@section{identification}%
2856     \bbl@exp{\bbl@inistore tag.ini=#1\\@@}%
2857     \bbl@inistore load.level=#2\\@@
2858     \bbl@loop@ini
2859     % == Process stored data ==
2860     \bbl@csarg\xdef{lini@\languagename}{#1}%
2861     \bbl@read@ini@aux
2862     % == 'Export' data ==
2863     \bbl@ini@exports{#2}%
2864     \global\bbl@csarg\let{inidata@\languagename}\bbl@inidata
2865     \global\let\bbl@inidata@empty
2866     \bbl@exp{\bbl@add@list\bbl@ini@loaded{\languagename}}%
2867     \bbl@tglobal\bbl@ini@loaded
2868     \fi
2869     \closein\bbl@readstream}
2870 \def\bbl@read@ini@aux{%
2871     \let\bbl@savestrings@empty
2872     \let\bbl@savetoday@empty
2873     \let\bbl@savodate@empty
2874     \def\bbl@elt##1##2##3{%
2875         \def\bbl@section{##1}%
2876         \in@{=date.}{=##1}% Find a better place
2877         \ifin@
2878             \bbl@ifunset{bbl@inikv@##1}%
2879             {\bbl@ini@calendar{##1}}%
2880             {}%
2881         \fi
2882         \bbl@ifunset{bbl@inikv@##1}{}%
2883         {\csname bbl@inikv@##1\endcsname{##2}{##3}}%
2884     \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.

```

2885 \def\bbl@extend@ini@aux#1{%
2886     \bbl@startcommands*{#1}{captions}%
2887     % Activate captions/... and modify exports
2888     \bbl@csarg\def{inikv@captions.licr}##1##2{%
2889         \setlocalecaption{#1}{##1}{##2}}%
2890     \def\bbl@inikv@captions##1##2{%
2891         \bbl@ini@captions@aux{##1}{##2}}%
2892     \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2893     \def\bbl@exportkey##1##2##3{%
2894         \bbl@ifunset{bbl@kv@##2}{%
2895             {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2896             \bbl@exp{\global\let<bbl@##1@\languagename>\<bbl@kv@##2>}}%
2897             \fi}}%
2898     % As with \bbl@read@ini, but with some changes

```

```

2899 \bbl@read@ini@aux
2900 \bbl@ini@exports\tw@
2901 % Update inidata@lang by pretending the ini is read.
2902 \def\bbl@elt##1##2##3{%
2903   \def\bbl@section{##1}%
2904   \bbl@iniline##2=##3\bbl@iniline}%
2905 \csname bbl@inidata@#1\endcsname
2906 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2907 \StartBabelCommands*{#1}{date}% And from the import stuff
2908 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2909 \bbl@savetoday
2910 \bbl@savedate
2911 \bbl@endcommands}

```

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

```

2912 \def\bbl@ini@calendar#1{%
2913   \lowercase{\def\bbl@tempa{=#1=}}%
2914   \bbl@replace\bbl@tempa{=date.gregorian}{}%
2915   \bbl@replace\bbl@tempa{=date.}{}%
2916   \in@{.licr}=#1=%
2917   \ifin@
2918     \ifcase\bbl@engine
2919       \bbl@replace\bbl@tempa{.licr}{}%
2920     \else
2921       \let\bbl@tempa\relax
2922     \fi
2923 \fi
2924 \ifx\bbl@tempa\relax\else
2925   \bbl@replace\bbl@tempa{=}{}%
2926   \ifx\bbl@tempa@empty\else
2927     \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2928   \fi
2929   \bbl@exp{%
2930     \def<\bbl@inikv@#1>#####1#####2{%
2931       \\bbl@inidate#####1...\relax{#####2}{\bbl@tempa}}%
2932 \fi}

```

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

```

2933 \def\bbl@renewinikey#1/#2\@@#3{%
2934   \edef\bbl@tempa{\zap@space #1 \@empty}% section
2935   \edef\bbl@tempb{\zap@space #2 \@empty}% key
2936   \bbl@trim\toks@{#3}% value
2937   \bbl@exp{%
2938     \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2939     \\g@addto@macro\\bbl@inidata{%
2940       \\bbl@elt{\bbl@tempa}{\bbl@tempb}{\the\toks@}}}%

```

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

```

2941 \def\bbl@exportkey#1#2#3{%
2942   \bbl@ifunset{\bbl@kv@#2}%
2943   {\bbl@csarg\gdef{#1@\languagename}{#3}}%
2944   {\expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
2945     \bbl@csarg\gdef{#1@\languagename}{#3}%
2946     \else
2947     \bbl@exp{\global\let<\bbl@#1@\languagename>\<\bbl@kv@#2>}%
2948     \fi}}

```

Key-value pairs are treated differently depending on the section in the ini file. The following macros are the readers for identification and typography. Note `\bbl@ini@exports` is called always (via `\bbl@inisec`), while `\bbl@after@ini` must be called explicitly after `\bbl@read@ini` if necessary.

Although BCP 47 doesn't treat '-x-' as an extension, the CLDR and many other sources do (as a *private use extension*). For consistency with other single-letter subtags or 'singletons', here is considered an extension, too.

```

2949 \def\bbl@iniwarning#1{%
2950   \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
2951     {\bbl@warning{%
2952       From babel-\bbl@cs{lini@\languagename}.ini:\\%
2953       \bbl@cs{@kv@identification.warning#1}\\%
2954       Reported }}}
2955 %
2956 \let\bbl@release@transforms@empty
2957 \let\bbl@release@casing@empty
2958 \def\bbl@ini@exports#1{%
2959   % Identification always exported
2960   \bbl@iniwarning{}%
2961   \ifcase\bbl@engine
2962     \bbl@iniwarning{.pdflatex}%
2963   \or
2964     \bbl@iniwarning{.lualatex}%
2965   \or
2966     \bbl@iniwarning{.xelatex}%
2967   \fi%
2968   \bbl@exportkey{llevel}{identification.load.level}{}%
2969   \bbl@exportkey{elname}{identification.name.english}{}%
2970   \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%
2971     {\csname bbl@elname@\languagename\endcsname}}%
2972   \bbl@exportkey{tbcpl}{identification.tag.bcp47}{}%
2973   % Somewhat hackish. TODO:
2974   \bbl@exportkey{casing}{identification.tag.bcp47}{}%
2975   \bbl@exportkey{lbcpl}{identification.language.tag.bcp47}{}%
2976   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2977   \bbl@exportkey{esname}{identification.script.name}{}%
2978   \bbl@exp{\\bbl@exportkey{sname}{identification.script.name.opentype}%
2979     {\csname bbl@esname@\languagename\endcsname}}%
2980   \bbl@exportkey{sbcpl}{identification.script.tag.bcp47}{}%
2981   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2982   \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2983   \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
2984   \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%
2985   \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
2986   \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
2987   % Also maps bcp47 -> languagename
2988   \ifbbl@bcptoname
2989     \bbl@csarg\xdef{bcp@map@\bbl@cl{tbcpl}}{\languagename}%
2990   \fi
2991   \ifcase\bbl@engine\or
2992     \directlua{%
2993       Babel.locale_props[\the\bbl@cs{id@\languagename}].script
2994       = '\bbl@cl{sbcpl}'}%
2995   \fi
2996   % Conditional
2997   \ifnum#1>\z@      % 0 = only info, 1, 2 = basic, (re)new
2998     \bbl@exportkey{calpr}{date.calendar.preferred}{}%
2999     \bbl@exportkey{lncr}{typography.linebreaking}{h}%
3000     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3001     \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
3002     \bbl@exportkey{rgtm}{typography.righthyphenmin}{3}%
3003     \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3004     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3005     \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3006     \bbl@exportkey{intsp}{typography.intraspaces}{u}%
3007     \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3008     \bbl@exportkey{chrng}{characters.ranges}{}%

```

```

3009 \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
3010 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3011 \ifnum#1=\tw@ % only (re)new
3012 \bbl@exportkey{rqtex}{identification.require.babel}{}%
3013 \bbl@tglobal\bbl@savetoday
3014 \bbl@tglobal\bbl@savestate
3015 \bbl@savestrings
3016 \fi
3017 \fi}

```

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

```

3018 \def\bbl@inikv#1#2{% key=value
3019 \toks@{#2}% This hides #'s from ini values
3020 \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}

```

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

```

3021 \let\bbl@inikv@identification\bbl@inikv
3022 \let\bbl@inikv@date\bbl@inikv
3023 \let\bbl@inikv@typography\bbl@inikv
3024 \let\bbl@inikv@numbers\bbl@inikv

```

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

```

3025 \def\bbl@maybextx{-\bbl@csarg\ifx{extx@\languagename}\@empty x-\fi}
3026 \def\bbl@inikv@characters#1#2{%
3027 \bbl@ifsamestring{#1}{casing}% eg, casing = uV
3028 {\bbl@exp{%
3029 \\\g@addto@macro\\\bbl@release@casing{%
3030 \\\bbl@casemapping}{\languagename}{\unexpanded{#2}}}}}%
3031 {\in@{#1}{#2}% eg, casing.Uv = uV
3032 \ifin@
3033 \lowercase{\def\bbl@tempb{#1}}%
3034 \bbl@replace\bbl@tempb{casing.}}}%
3035 \bbl@exp{\\g@addto@macro\\\bbl@release@casing{%
3036 \\\bbl@casemapping
3037 {\\\bbl@maybextx\bbl@tempb}{\languagename}{\unexpanded{#2}}}}}%
3038 \else
3039 \bbl@inikv{#1}{#2}%
3040 \fi}}

```

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

```

3041 \def\bbl@inikv@counters#1#2{%
3042 \bbl@ifsamestring{#1}{digits}%
3043 {\bbl@error{The counter name 'digits' is reserved for mapping\\%
3044 decimal digits}%
3045 {Use another name.}}}%
3046 {}%
3047 \def\bbl@tempc{#1}%
3048 \bbl@trim@def{\bbl@tempb*}{#2}%
3049 \in@{.1}{#1}%
3050 \ifin@
3051 \bbl@replace\bbl@tempc{.1}{}%
3052 \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\languagename}{%
3053 \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3054 \fi
3055 \in@{.F}{#1}%
3056 \ifin@ \else \in@{.S}{#1} \fi
3057 \ifin@
3058 \bbl@csarg\protected@xdef{cntr@#1@\languagename}{\bbl@tempb*}%
3059 \else
3060 \toks@{}% Required by \bbl@builddifcase, which returns \bbl@tempa

```

```

3061 \expandafter\bb@buildifcase\bb@tempb* \ % Space after \
3062 \bb@csarg{\global\expandafter\let}{cntr@#1@\languagename}\bb@tempa
3063 \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:

```

3064 \ifcase\bb@engine
3065 \bb@csarg\def{inikv@captions.licr}#1#2{%
3066 \bb@ini@captions@aux{#1}{#2}}
3067 \else
3068 \def\bb@inikv@captions#1#2{%
3069 \bb@ini@captions@aux{#1}{#2}}
3070 \fi

```

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

```

3071 \def\bb@ini@captions@template#1#2{% string language tempa=capt-name
3072 \bb@replace\bb@tempa{.template}}%
3073 \def\bb@toreplace{#1}{}%
3074 \bb@replace\bb@toreplace{[ ]}{\nobreakspace}}%
3075 \bb@replace\bb@toreplace{[ ]}{\csname}%
3076 \bb@replace\bb@toreplace{[ ]}{\csname the}%
3077 \bb@replace\bb@toreplace{[ ]}{\name\endcsname}}%
3078 \bb@replace\bb@toreplace{[ ]}{\endcsname}}%
3079 \bb@xin@{,\bb@tempa,}{,chapter,appendix,part,}%
3080 \ifin@
3081 \@nameuse{\bb@patch\bb@tempa}%
3082 \global\bb@csarg\let{\bb@tempa fmt@#2}\bb@toreplace
3083 \fi
3084 \bb@xin@{,\bb@tempa,}{,figure,table,}%
3085 \ifin@
3086 \global\bb@csarg\let{\bb@tempa fmt@#2}\bb@toreplace
3087 \bb@exp{\gdef\<fnum@\bb@tempa>{%
3088 \\\bb@ifunset{\bb@\bb@tempa fmt@\\\languagename}%
3089 {\[fnum@\bb@tempa]}%
3090 {\\\@nameuse{\bb@\bb@tempa fmt@\\\languagename}}}}%
3091 \fi}
3092 \def\bb@ini@captions@aux#1#2{%
3093 \bb@trim@def\bb@tempa{#1}%
3094 \bb@xin@{.template}{\bb@tempa}%
3095 \ifin@
3096 \bb@ini@captions@template{#2}\languagename
3097 \else
3098 \bb@ifblank{#2}%
3099 {\bb@exp{%
3100 \toks@{\\\bb@nocaption{\bb@tempa}{\languagename\bb@tempa name}}}}%
3101 {\bb@trim\toks@{#2}}%
3102 \bb@exp{%
3103 \\\bb@add\\\bb@savestrings{%
3104 \\\SetString\<\bb@tempa name>{\the\toks@}}}%
3105 \toks@\expandafter{\bb@captionslist}%
3106 \bb@exp{\\\in@{\<\bb@tempa name>}{\the\toks@}}%
3107 \ifin@\else
3108 \bb@exp{%
3109 \\\bb@add\<\bb@extracaps@\languagename>{\<\bb@tempa name>}%
3110 \\\bb@toglobal\<\bb@extracaps@\languagename>}}%
3111 \fi
3112 \fi}

```

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

```

3113 \def\bb@list@the{%
3114 part,chapter,section,subsection,subsubsection,paragraph,%
3115 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3116 table,page,footnote,mpfootnote,mpfn}

```

```

3117 \def\bbL@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3118 \bbL@ifunset{\bbL@map@#1@languagename}%
3119   {\@nameuse{#1}}%
3120   {\@nameuse{\bbL@map@#1@languagename}}%
3121 \def\bbL@inikv@labels#1#2{%
3122   \in@{.map}{#1}%
3123   \ifin@
3124     \ifx\bbL@KVP@labels\@nnil\else
3125       \bbL@xin@{ map }{ \bbL@KVP@labels\space}%
3126       \ifin@
3127         \def\bbL@tempc{#1}%
3128         \bbL@replace\bbL@tempc{.map}{}%
3129         \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3130         \bbL@exp{%
3131           \gdef<\bbL@map@\bbL@tempc @languagename>%
3132             {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3133         \bbL@foreach\bbL@list@the{%
3134           \bbL@ifunset{the##1}{}%
3135           {\bbL@exp{\let\\bbL@tempd<the##1>}%
3136             \bbL@exp{%
3137               \\bbL@sreplace<the##1>%
3138               {\<\bbL@tempc>{##1}}{\\bbL@map@cnt{\bbL@tempc}{##1}}%
3139               \\bbL@sreplace<the##1>%
3140               {\<\empty @\bbL@tempc><c@##1>}{\\bbL@map@cnt{\bbL@tempc}{##1}}}%
3141           \expandafter\ifx\csname the##1\endcsname\bbL@tempd\else
3142             \toks@\expandafter\expandafter\expandafter{%
3143               \csname the##1\endcsname}%
3144             \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3145             \fi}}%
3146     \fi
3147   \fi
3148   %
3149 \else
3150   %
3151   % The following code is still under study. You can test it and make
3152   % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3153   % language dependent.
3154   \in@{enumerate.}{#1}%
3155   \ifin@
3156     \def\bbL@tempa{#1}%
3157     \bbL@replace\bbL@tempa{enumerate.}{}%
3158     \def\bbL@toreplace{#2}%
3159     \bbL@replace\bbL@toreplace{[ ]}{\nobreakspace{}}%
3160     \bbL@replace\bbL@toreplace{[ ]}{\csname the}%
3161     \bbL@replace\bbL@toreplace{[ ]}{\endcsname}}%
3162     \toks@\expandafter{\bbL@toreplace}%
3163     % TODO. Execute only once:
3164     \bbL@exp{%
3165       \\bbL@add<extras\languagename>%
3166       \\babel@save<labelenum\romannumeral\bbL@tempa>%
3167       \def<labelenum\romannumeral\bbL@tempa>{\the\toks@}}%
3168       \\bbL@toggle\languagename>%
3169     \fi
3170 \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.

```

3171 \def\bbL@chapttype{chapter}
3172 \ifx\@makechapterhead\@undefined
3173   \let\bbL@patchchapter\relax
3174 \else\ifx\thechapter\@undefined

```

```

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

```

3207 \let\bbbl@calendar\@empty
3208 \DeclareRobustCommand\localdate[1][\bbbl@localdate{#1}]
3209 \def\bbbl@localdate#1#2#3#4{%
3210 \begingroup
3211 \edef\bbbl@they{#2}%
3212 \edef\bbbl@them{#3}%
3213 \edef\bbbl@thed{#4}%
3214 \edef\bbbl@tempe{%
3215 \bbbl@ifunset{bbbl@calpr@\languagename}{\bbbl@cl{calpr}},%
3216 #1}%
3217 \bbbl@replace\bbbl@tempe{ }{}%
3218 \bbbl@replace\bbbl@tempe{CONVERT}{convert=}% Hackish
3219 \bbbl@replace\bbbl@tempe{convert}{convert=}%
3220 \let\bbbl@ld@calendar\@empty
3221 \let\bbbl@ld@variant\@empty
3222 \let\bbbl@ld@convert\relax
3223 \def\bbbl@tempb##1=##2\@@{\@namedef{bbbl@ld@##1}{##2}}%
3224 \bbbl@foreach\bbbl@tempe{\bbbl@tempb##1\@@}%
3225 \bbbl@replace\bbbl@ld@calendar{gregorian}{}%
3226 \ifx\bbbl@ld@calendar\@empty\else
3227 \ifx\bbbl@ld@convert\relax\else
3228 \babelcalendar[\bbbl@they-\bbbl@them-\bbbl@thed]%
3229 {\bbbl@ld@calendar}\bbbl@they\bbbl@them\bbbl@thed
3230 \fi
3231 \fi
3232 \@nameuse{bbbl@precalendar}% Remove, eg, +, -civil (-ca-islamic)
3233 \edef\bbbl@calendar{% Used in \month..., too
3234 \bbbl@ld@calendar

```

```

3235     \ifx\bbld@variant\@empty\else
3236     .\bbld@variant
3237     \fi}%
3238     \bbld@cased
3239     {\@nameuse{bbld@date@\languagename @\bbld@calendar}%
3240     \bbld@they\bbld@them\bbld@thed}%
3241 \endgroup}
3242 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3243 \def\bbld@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3244 \bbld@trim@def\bbld@tempa{#1.#2}%
3245 \bbld@ifsamestring{\bbld@tempa}{months.wide}%      to savedate
3246 {\bbld@trim@def\bbld@tempa{#3}%
3247 \bbld@trim\toks@{#5}%
3248 \@temptokena\expandafter{\bbld@savedate}%
3249 \bbld@exp{% Reverse order - in ini last wins
3250 \def\\bbld@savedate{%
3251 \\SetString<month\romannumeral\bbld@tempa#6name>{\the\toks@}%
3252 \the\@temptokena}}}%
3253 {\bbld@ifsamestring{\bbld@tempa}{date.long}%      defined now
3254 {\lowercase{\def\bbld@tempb{#6}}}%
3255 \bbld@trim@def\bbld@toreplace{#5}%
3256 \bbld@TG@@date
3257 \global\bbld@csarg\let{date@\languagename @\bbld@tempb}\bbld@toreplace
3258 \ifx\bbld@savetoday\@empty
3259 \bbld@exp{% TODO. Move to a better place.
3260 \\AfterBabelCommands{%
3261 \def<\languagename date>{\\protect<\languagename date >}%
3262 \\newcommand<\languagename date >[4][[%
3263 \\bbld@usedategrouprue
3264 <\bbld@ensure@\languagename>{%
3265 \\localdate[####1]{####2}{####3}{####4}}}%
3266 \def\\bbld@savetoday{%
3267 \\SetString\\today{%
3268 <\languagename date>[convert]%
3269 {\\the\year}{\\the\month}{\\the\day}}}%
3270 \fi}%
3271 {}}}

```

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 `\bbld@replace\toks@` contains the resulting string, which is used by `\bbld@replace@finish@iii` (this implicit behavior doesn’t seem a good idea, but it’s efficient).

```

3272 \let\bbld@calendar\@empty
3273 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3274 \@nameuse{bbld@ca@#2}#1\@}
3275 \newcommand\babelDateSpace{\nobreakspace}
3276 \newcommand\babelDateDot{.\@} % TODO. \let instead of repeating
3277 \newcommand\babelDated[1]{\number#1}
3278 \newcommand\babelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3279 \newcommand\babelDateM[1]{\number#1}
3280 \newcommand\babelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3281 \newcommand\babelDateMMM[1]{%
3282 \csname month\romannumeral#1\bbld@calendar name\endcsname}%
3283 \newcommand\babelDatey[1]{\number#1}%
3284 \newcommand\babelDateyy[1]{%
3285 \ifnum#1<10 0\number#1 %
3286 \else\ifnum#1<100 \number#1 %
3287 \else\ifnum#1<1000 \expandafter@gobble\number#1 %
3288 \else\ifnum#1<10000 \expandafter@gobbletwo\number#1 %
3289 \else
3290 \bbld@error
3291 {Currently two-digit years are restricted to the\

```



```

3292     range 0-9999.)%
3293     {There is little you can do. Sorry.}%
3294     \fi\fi\fi\fi}}
3295 \newcommand\BabelDateyyy[1]{\number#1} % TODO - add leading 0
3296 \newcommand\BabelDateU[1]{\number#1}%
3297 \def\bbl@replace@finish@iii#1{%
3298   \bbl@exp{\def\#1###1###2###3{\the\toks@}}
3299 \def\bbl@TG@@date{%
3300   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3301   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3302   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{###3}}%
3303   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{###3}}%
3304   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{###2}}%
3305   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{###2}}%
3306   \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{###2}}%
3307   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{###1}}%
3308   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{###1}}%
3309   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{###1}}%
3310   \bbl@replace\bbl@toreplace{[U]}{\BabelDateU{###1}}%
3311   \bbl@replace\bbl@toreplace{[y|]}{\bbl@datecntr[###1|]}%
3312   \bbl@replace\bbl@toreplace{[U|]}{\bbl@datecntr[###1|]}%
3313   \bbl@replace\bbl@toreplace{[m|]}{\bbl@datecntr[###2|]}%
3314   \bbl@replace\bbl@toreplace{[d|]}{\bbl@datecntr[###3|]}%
3315   \bbl@replace@finish@iii\bbl@toreplace}
3316 \def\bbl@datecntr{\expandafter\bbl@xdatecntr\expandafter}
3317 \def\bbl@xdatecntr[#1|#2]{\localenumeral{#2}{#1}}

```

Transforms.

```

3318 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3319 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3320 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3321   #1[#2]{#3}{#4}{#5}}
3322 \begingroup % A hack. TODO. Don't require an specific order
3323   \catcode`\%=12
3324   \catcode`\&=14
3325   \gdef\bbl@transforms#1#2#3{&%
3326     \directlua{
3327       local str = [==[#2]==]
3328       str = str:gsub('%.%d+%.%d+$', '')
3329       token.set_macro('babeltempa', str)
3330     }&%
3331     \def\babeltempc{ }&%
3332     \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
3333     \ifin@\else
3334       \bbl@xin@{: \babeltempa,}{,\bbl@KVP@transforms,}&%
3335     \fi
3336     \ifin@
3337       \bbl@foreach\bbl@KVP@transforms{&%
3338         \bbl@xin@{: \babeltempa,}{,##1,}&%
3339         \ifin@ &% font:font:transform syntax
3340         \directlua{
3341           local t = {}
3342           for m in string.gmatch('##1'..':', '(.-):') do
3343             table.insert(t, m)
3344           end
3345           table.remove(t)
3346           token.set_macro('babeltempc', ',fonts=' .. table.concat(t, ' '))
3347         }&%
3348         \fi}&%
3349     \in@{.0$}{#2$}&%
3350     \ifin@
3351       \directlua{&% (\attribute) syntax
3352         local str = string.match([[ \bbl@KVP@transforms]],

```

```

3353             '%([[^%(-)^%][^%)]-\babeltempa')
3354         if str == nil then
3355             token.set_macro('babeltempb', '')
3356         else
3357             token.set_macro('babeltempb', ',attribute=' .. str)
3358         end
3359     }&%
3360     \toks@{#3}&%
3361     \bbl@exp{&%
3362         \\g@addto@macro\\bbl@release@transforms{&%
3363             \relax &% Closes previous \bbl@transforms@aux
3364             \\bbl@transforms@aux
3365             \\#1{label=\babeltempa\babeltempb\babeltempc}&%
3366             {\languagename}{\the\toks@}}&%
3367     \else
3368         \g@addto@macro\bbl@release@transforms{, {#3}}&%
3369     \fi
3370 \fi}
3371 \endgroup

```

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

```

3372 \def\bbl@provide@lsys#1{%
3373     \bbl@ifunset{bbl@lname@#1}%
3374     {\bbl@load@info{#1}}%
3375     }%
3376     \bbl@csarg\let{lsys@#1}\@empty
3377     \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3378     \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3379     \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3380     \bbl@ifunset{bbl@lname@#1}{%
3381         {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3382     \ifcase\bbl@engine\or\or
3383         \bbl@ifunset{bbl@prehc@#1}{%
3384             {\bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3385             }%
3386             {\ifx\bbl@xenoxyph\undefined
3387                 \global\let\bbl@xenoxyph\bbl@xenoxyph@d
3388                 \ifx\AtBeginDocument\@notprerr
3389                     \expandafter\@secondoftwo % to execute right now
3390                 \fi
3391                 \AtBeginDocument{%
3392                     \bbl@patchfont{\bbl@xenoxyph}%
3393                     {\expandafter\select@language\expandafter{\languagename}}}%
3394                 \fi}}%
3395     \fi
3396     \bbl@csarg\bbl@tglobal{lsys@#1}}
3397 \def\bbl@xenoxyph@d{%
3398     \bbl@ifset{bbl@prehc@\languagename}%
3399     {\ifnum\hyphenchar\font=\defaultthyphenchar
3400         \iffontchar\font\bbl@cl{prehc}\relax
3401         \hyphenchar\font\bbl@cl{prehc}\relax
3402     \else\iffontchar\font"200B
3403         \hyphenchar\font"200B
3404     \else
3405         \bbl@warning
3406         {Neither 0 nor ZERO WIDTH SPACE are available\\%
3407         in the current font, and therefore the hyphen\\%
3408         will be printed. Try changing the fontspec's\\%
3409         'HyphenChar' to another value, but be aware\\%
3410         this setting is not safe (see the manual).\\%
3411         Reported}%
3412         \hyphenchar\font\defaultthyphenchar

```

```

3413     \fi\fi
3414     \fi}%
3415     {\hyphenchar\font\defaultthyphenchar}}
3416     % \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).

```

3417 \def\babel@load@info#1{%
3418   \def\BabelBeforeIni##1##2{%
3419     \begingroup
3420     \babel@read@ini{##1}0%
3421     \endinput           % babel- .tex may contain onlypreamble's
3422     \endgroup}%       boxed, to avoid extra spaces:
3423   {\babel@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.

```

3424 \def\babel@setdigits#1#2#3#4#5{%
3425   \babel@exp{%
3426     \def<\languagename digits>####1{%       ie, \langdigits
3427       <\babel@digits@\languagename>####1\\\@nil}%
3428     \let<\babel@cntr@digits@\languagename><\languagename digits>%
3429     \def<\languagename counter>####1{%       ie, \langcounter
3430       \\expandafter<\babel@counter@\languagename>%
3431       \\csname c@####1\endcsname}%
3432     \def<\babel@counter@\languagename>####1{% ie, \babel@counter@lang
3433       \\expandafter<\babel@digits@\languagename>%
3434       \\number####1\\\@nil}}%
3435   \def\babel@tempa##1##2##3##4##5{%
3436     \babel@exp{%   Wow, quite a lot of hashes! :- (
3437       \def<\babel@digits@\languagename>#####1{%
3438         \\ifx#####1\\\@nil           % ie, \babel@digits@lang
3439         \\else
3440           \\ifx0#####1#1%
3441           \\else\\ifx1#####1#2%
3442           \\else\\ifx2#####1#3%
3443           \\else\\ifx3#####1#4%
3444           \\else\\ifx4#####1#5%
3445           \\else\\ifx5#####1##1%
3446           \\else\\ifx6#####1##2%
3447           \\else\\ifx7#####1##3%
3448           \\else\\ifx8#####1##4%
3449           \\else\\ifx9#####1##5%
3450           \\else#####1%
3451           \\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi\\fi
3452           \\expandafter<\babel@digits@\languagename>%
3453           \\fi}}}%
3454   \babel@tempa}

```

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

```

3455 \def\babel@buildifcase#1 {% Returns \babel@tempa, requires \toks@={ }
3456   \ifx\\#1%           % \\ before, in case #1 is multiletter
3457     \babel@exp{%
3458       \def\\babel@tempa####1{%
3459         <ifcase>####1\space\the\toks@<else>\\\@ctrerr<fi>}}%
3460     \else
3461       \toks@\\expandafter{\the\toks@<or #1>%
3462       \expandafter\babel@buildifcase
3463     \fi}

```

The code for additive counters is somewhat tricky and it's based on the fact the arguments just before @@ collects digits which have been left ‘unused’ in previous arguments, the first of them

being the number of digits in the number to be converted. This explains the reverse set 76543210. Digits above 10000 are not handled yet. When the key contains the subkey .F., the number after is treated as an special case, for a fixed form (see babel-he.ini, for example).

```

3464 \newcommand\localenumerical[2]{\bbl@cs{cntr@#1@ \language name}{#2}}
3465 \def\bbl@localecntr#1#2{\localenumerical{#2}{#1}}
3466 \newcommand\localecounter[2]{%
3467   \expandafter\bbl@localecntr
3468   \expandafter{\number\csname c@#2\endcsname}{#1}}
3469 \def\bbl@alphanumeric#1#2{%
3470   \expandafter\bbl@alphanumeric@i\number#2 76543210\@@{#1}}
3471 \def\bbl@alphanumeric@i#1#2#3#4#5#6#7#8\@#9{%
3472   \ifcase@car#8\@nil\or   % Currently <10000, but prepared for bigger
3473     \bbl@alphanumeric@ii{#9}000000#1\or
3474     \bbl@alphanumeric@ii{#9}00000#1#2\or
3475     \bbl@alphanumeric@ii{#9}0000#1#2#3\or
3476     \bbl@alphanumeric@ii{#9}000#1#2#3#4\else
3477     \bbl@alphanum@invalid{>9999}%
3478   \fi}
3479 \def\bbl@alphanumeric@ii#1#2#3#4#5#6#7#8{%
3480   \bbl@ifunset{\bbl@cntr@#1.F.\number#5#6#7#8@ \language name}%
3481     {\bbl@cs{cntr@#1.4@ \language name}#5%
3482      \bbl@cs{cntr@#1.3@ \language name}#6%
3483      \bbl@cs{cntr@#1.2@ \language name}#7%
3484      \bbl@cs{cntr@#1.1@ \language name}#8%
3485      \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3486        \bbl@ifunset{\bbl@cntr@#1.S.321@ \language name}{}}%
3487        {\bbl@cs{cntr@#1.S.321@ \language name}}%
3488      \fi}%
3489   {\bbl@cs{cntr@#1.F.\number#5#6#7#8@ \language name}}}}
3490 \def\bbl@alphanum@invalid#1{%
3491   \bbl@error{Alphabetic numeral too large (#1)}%
3492   {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.

```

3493 \def\bbl@localeinfo#1#2{%
3494   \bbl@ifunset{\bbl@info@#2}{#1}%
3495     {\bbl@ifunset{\bbl@ \csname bbl@info@#2\endcsname @ \language name}{#1}%
3496      {\bbl@cs{\csname bbl@info@#2\endcsname @ \language name}}}}
3497 \newcommand\localeinfo[1]{%
3498   \ifx*#1\@empty   % TODO. A bit hackish to make it expandable.
3499     \bbl@afterelse\bbl@localeinfo{}%
3500   \else
3501     \bbl@localeinfo
3502     {\bbl@error{I've found no info for the current locale.\%
3503               The corresponding ini file has not been loaded\%
3504               Perhaps it doesn't exist}%
3505      {See the manual for details.}}%
3506     {#1}%
3507   \fi}
3508 % \@namedef{\bbl@info@name.locale}{\lcname}
3509 \@namedef{\bbl@info@tag.ini}{\lini}
3510 \@namedef{\bbl@info@name.english}{\elname}
3511 \@namedef{\bbl@info@name.opentype}{\lname}
3512 \@namedef{\bbl@info@tag.bcp47}{\tbc}
3513 \@namedef{\bbl@info@language.tag.bcp47}{\lbc}
3514 \@namedef{\bbl@info@tag.opentype}{\lotf}
3515 \@namedef{\bbl@info@script.name}{\esname}
3516 \@namedef{\bbl@info@script.name.opentype}{\sname}
3517 \@namedef{\bbl@info@script.tag.bcp47}{\sbcp}
3518 \@namedef{\bbl@info@script.tag.opentype}{\sotf}
3519 \@namedef{\bbl@info@region.tag.bcp47}{\rbcp}
3520 \@namedef{\bbl@info@variant.tag.bcp47}{\vbc}

```

```

3521 \@namedef{bbl@info@extension.t.tag.bcp47}{extt}
3522 \@namedef{bbl@info@extension.u.tag.bcp47}{extu}
3523 \@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.

```

3524 \ifcase\bbl@engine % Converts utf8 to its code (expandable)
3525 \def\bbl@outftocode#1{\the\numexpr\decode@UTFviii#1\relax}
3526 \else
3527 \def\bbl@outftocode#1{\expandafter`\string#1}
3528 \fi
3529 % Still somewhat hackish. WIP. Note |\str_if_eq:nnTF| is fully
3530 % expandable (|\bbl@ifsamestring| isn't).
3531 \providecommand\BCPdata{}
3532 \ifx\renewcommand\undefined\else % For plain. TODO. It's a quick fix
3533 \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty}
3534 \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3535 \nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3536 {\bbl@bcpdata@ii{#6}\bbl@main@language}%
3537 {\bbl@bcpdata@ii{#1#2#3#4#5#6}\languagename}}%
3538 \def\bbl@bcpdata@ii#1#2{%
3539 \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3540 {\bbl@error{Unknown field '#1' in \string\BCPdata.\.%
3541 Perhaps you misspelled it.}%
3542 {See the manual for details.}}%
3543 {\bbl@ifunset{bbl@csname bbl@info@#1.tag.bcp47\endcsname @#2}{}%
3544 {\bbl@cs{csname bbl@info@#1.tag.bcp47\endcsname @#2}}}%
3545 \fi
3546 \@namedef{bbl@info@casing.tag.bcp47}{casing}
3547 \newcommand\BabelUppercaseMapping[3]{%
3548 \DeclareUppercaseMapping[\nameuse{bbl@casing@#1}]{#2}{#3}}
3549 \newcommand\BabelTitlecaseMapping[3]{%
3550 \DeclareTitlecaseMapping[\nameuse{bbl@casing@#1}]{#2}{#3}}
3551 \newcommand\BabelLowercaseMapping[3]{%
3552 \DeclareLowercaseMapping[\nameuse{bbl@casing@#1}]{#2}{#3}}

```

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

```

3553 \def\bbl@casemapping#1#2#3{% 1:variant
3554 \def\bbl@tempa##1 ##2{% Loop
3555 \bbl@casemapping@i{##1}%
3556 \ifx\@empty##2\else\bbl@afterfi\bbl@tempa##2\fi}%
3557 \edef\bbl@templ{\nameuse{bbl@casing@#2}#1}% Language code
3558 \def\bbl@tempe{0}% Mode (upper/lower...)
3559 \def\bbl@tempc{#3}% Casing list
3560 \expandafter\bbl@tempa\bbl@tempc\@empty}
3561 \def\bbl@casemapping@i#1{%
3562 \def\bbl@tempb{#1}%
3563 \ifcase\bbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
3564 \nameuse{regex_replace_all:nnN}%
3565 {[{\x{c0}-\x{ff}][\x{80}-\x{bf}]*}{\0}}\bbl@tempb
3566 \else
3567 \nameuse{regex_replace_all:nnN}{.}{\0}}\bbl@tempb % TODO. needed?
3568 \fi
3569 \expandafter\bbl@casemapping@ii\bbl@tempb\@}
3570 \def\bbl@casemapping@ii#1#2#3\@{%
3571 \in@{#1#3}{<>}% ie, if <u>, <l>, <t>
3572 \ifin@
3573 \edef\bbl@tempe{%
3574 \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
3575 \else
3576 \ifcase\bbl@tempe\relax
3577 \DeclareUppercaseMapping[\bbl@templ]{\bbl@outftocode{#1}}{#2}%
3578 \DeclareLowercaseMapping[\bbl@templ]{\bbl@outftocode{#2}}{#1}%

```

```

3579 \or
3580 \DeclareUppercaseMapping[\bbl@templ]{\bbl@utfcode{#1}}{#2}%
3581 \or
3582 \DeclareLowercaseMapping[\bbl@templ]{\bbl@utfcode{#1}}{#2}%
3583 \or
3584 \DeclareTitlecaseMapping[\bbl@templ]{\bbl@utfcode{#1}}{#2}%
3585 \fi
3586 \fi}

```

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

```

3587 <<(*More package options)>> ≡
3588 \DeclareOption{ensureinfo=off}{}
3589 <</More package options>>
3590 \let\bbl@ensureinfo@gobble
3591 \newcommand\BabelEnsureInfo{%
3592 \ifx\InputIfFileExists\@undefined\else
3593 \def\bbl@ensureinfo##1{%
3594 \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3595 \fi
3596 \bbl@foreach\bbl@loaded{{%
3597 \let\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files
3598 \def\language##1{%
3599 \bbl@ensureinfo{##1}}}}
3600 \@ifpackagewith{babel}{ensureinfo=off}{}%
3601 {\AtEndOfPackage{% Test for plain.
3602 \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`.

```

3603 \newcommand\getlocaleproperty{%
3604 \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3605 \def\bbl@getproperty@s#1#2#3{%
3606 \let#1\relax
3607 \def\bbl@elt##1##2##3{%
3608 \bbl@ifsamestring{##1/##2}{#3}%
3609 {\providecommand#1{##3}%
3610 \def\bbl@elt####1####2####3{}}}%
3611 {}}%
3612 \bbl@cs{inidata@#2}}%
3613 \def\bbl@getproperty@x#1#2#3{%
3614 \bbl@getproperty@s{#1}{#2}{#3}%
3615 \ifx#1\relax
3616 \bbl@error
3617 {Unknown key for locale '#2':\%
3618 #3\%
3619 \string#1 will be set to \relax}%
3620 {Perhaps you misspelled it.}%
3621 \fi}
3622 \let\bbl@ini@loaded\@empty
3623 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
3624 \def\ShowLocaleProperties#1{%
3625 \typeout{}}%
3626 \typeout{*** Properties for language '#1' ***}
3627 \def\bbl@elt##1##2##3{\typeout{##1/##2 = ##3}}%
3628 \@nameuse{bbl@inidata@#1}%
3629 \typeout{*****}}

```

5 Adjusting the Babel behavior

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

```

3630 \newcommand\babeladjust[1]{% TODO. Error handling.

```

```

3631 \bbl@forkv{#1}{%
3632   \bbl@ifunset{bbl@ADJ@##1@##2}%
3633     {\bbl@cs{ADJ@##1}{##2}}%
3634     {\bbl@cs{ADJ@##1@##2}}}
3635 %
3636 \def\bbl@adjust@lua#1#2{%
3637   \ifvmode
3638     \ifnum\currentgrouplevel=\z@
3639       \directlua{ Babe\.#2 }%
3640       \expandafter\expandafter\expandafter\@gobble
3641       \fi
3642   \fi
3643   {\bbl@error   % The error is gobbled if everything went ok.
3644     {Currently, #1 related features can be adjusted only\\%
3645       in the main vertical list.}%
3646     {Maybe things change in the future, but this is what it is.}}}
3647 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3648   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3649 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3650   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3651 \@namedef{bbl@ADJ@bidi.text@on}{%
3652   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3653 \@namedef{bbl@ADJ@bidi.text@off}{%
3654   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3655 \@namedef{bbl@ADJ@bidi.math@on}{%
3656   \let\bbl@noamsmath\@empty}
3657 \@namedef{bbl@ADJ@bidi.math@off}{%
3658   \let\bbl@noamsmath\relax}
3659 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3660   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3661 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3662   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3663 %
3664 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3665   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3666 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3667   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3668 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3669   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3670 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3671   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3672 \@namedef{bbl@ADJ@justify.arabic@on}{%
3673   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3674 \@namedef{bbl@ADJ@justify.arabic@off}{%
3675   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3676 %
3677 \def\bbl@adjust@layout#1{%
3678   \ifvmode
3679     #1%
3680     \expandafter\@gobble
3681     \fi
3682   {\bbl@error   % The error is gobbled if everything went ok.
3683     {Currently, layout related features can be adjusted only\\%
3684       in vertical mode.}%
3685     {Maybe things change in the future, but this is what it is.}}}
3686 \@namedef{bbl@ADJ@layout.tabular@on}{%
3687   \ifnum\bbl@tabular@mode=\tw@
3688     \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}%
3689   \else
3690     \chardef\bbl@tabular@mode\@ne
3691   \fi}
3692 \@namedef{bbl@ADJ@layout.tabular@off}{%
3693   \ifnum\bbl@tabular@mode=\tw@

```

```

3694 \bbl@adjust@layout{\let\@tabular\bbl@0L@tabular}%
3695 \else
3696 \chardef\bbl@tabular@mode\z@
3697 \fi}
3698 \@namedef{bbl@ADJ@layout.lists@on}{%
3699 \bbl@adjust@layout{\let\list\bbl@NL@list}}
3700 \@namedef{bbl@ADJ@layout.lists@off}{%
3701 \bbl@adjust@layout{\let\list\bbl@0L@list}}
3702 %
3703 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3704 \bbl@bcppallowedtrue}
3705 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3706 \bbl@bcppallowedfalse}
3707 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3708 \def\bbl@bcp@prefix{#1}}
3709 \def\bbl@bcp@prefix{bcp47-}
3710 \@namedef{bbl@ADJ@autoload.options}#1{%
3711 \def\bbl@autoload@options{#1}}
3712 \let\bbl@autoload@bcptoptions\@empty
3713 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3714 \def\bbl@autoload@bcptoptions{#1}}
3715 \newif\ifbbl@bcptoname
3716 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3717 \bbl@bcptonametrue}
3718 \BabelEnsureInfo}
3719 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3720 \bbl@bcptonamefalse}
3721 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3722 \directlua{ Babel.ignore_pre_char = function(node)
3723 return (node.lang == \the\csname l@nohyphenation\endcsname)
3724 end }}
3725 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3726 \directlua{ Babel.ignore_pre_char = function(node)
3727 return false
3728 end }}
3729 \@namedef{bbl@ADJ@select.write@shift}{%
3730 \let\bbl@restorelastskip\relax
3731 \def\bbl@savelastskip{%
3732 \let\bbl@restorelastskip\relax
3733 \ifvmode
3734 \ifdim\lastskip=\z@
3735 \let\bbl@restorelastskip\nobreak
3736 \else
3737 \bbl@exp{%
3738 \def\\bbl@restorelastskip{%
3739 \skip@=\the\lastskip
3740 \\nobreak \vskip-\skip@ \vskip\skip@}}%
3741 \fi
3742 \fi}}
3743 \@namedef{bbl@ADJ@select.write@keep}{%
3744 \let\bbl@restorelastskip\relax
3745 \let\bbl@savelastskip\relax}
3746 \@namedef{bbl@ADJ@select.write@omit}{%
3747 \AddBabelHook{babel-select}{beforestart}{%
3748 \expandafter\babel@aux\expandafter{\bbl@main@language}}}%
3749 \let\bbl@restorelastskip\relax
3750 \def\bbl@savelastskip##1\bbl@restorelastskip}}
3751 \@namedef{bbl@ADJ@select.encoding@off}{%
3752 \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.

```

3753 <<(*More package options)>> ≡
3754 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3755 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3756 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3757 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3758 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3759 <</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.

```

3760 \bbl@trace{Cross referencing macros}
3761 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3762   \def\@newl@bel#1#2#3{%
3763     \@safe@activestrue
3764     \bbl@ifunset{#1@#2}%
3765     \relax
3766     {\gdef\@multiplelabels{%
3767       \@latex@warning@no@line{There were multiply-defined labels}}%
3768     \@latex@warning@no@line{Label `#2' multiply defined}}%
3769     \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.

```

3770 \CheckCommand*\@testdef[3]{%
3771   \def\reserved@a{#3}%
3772   \expandafter\ifx\c@name#1@#2\endcsname\reserved@a
3773   \else
3774     \@tempwattrue
3775   \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.

```

3776 \def\@testdef#1#2#3{% TODO. With @samestring?
3777   \@safe@activestrue
3778   \expandafter\let\expandafter\bbl@tempa\c@name #1@#2\endcsname
3779   \def\bbl@tempb{#3}%
3780   \@safe@activesfalse
3781   \ifx\bbl@tempa\relax
3782   \else
3783     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3784   \fi
3785   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3786   \ifx\bbl@tempa\bbl@tempb
3787   \else
3788     \@tempwattrue
3789   \fi}
3790 \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.

```

3791 \bbl@xin@{R}\bbl@opt@safe
3792 \ifin@
3793 \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3794 \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3795 {\expandafter\strip@prefix\meaning\ref}%
3796 \ifin@
3797 \bbl@redefine\@kernel@ref#1{%
3798 \@safe@activestruel\org@@kernel@ref{#1}\@safe@activesfalse}
3799 \bbl@redefine\@kernel@pageref#1{%
3800 \@safe@activestruel\org@@kernel@pageref{#1}\@safe@activesfalse}
3801 \bbl@redefine\@kernel@sref#1{%
3802 \@safe@activestruel\org@@kernel@sref{#1}\@safe@activesfalse}
3803 \bbl@redefine\@kernel@spageref#1{%
3804 \@safe@activestruel\org@@kernel@spageref{#1}\@safe@activesfalse}
3805 \else
3806 \bbl@redefinero\ref#1{%
3807 \@safe@activestruel\org@ref{#1}\@safe@activesfalse}
3808 \bbl@redefinero\pageref#1{%
3809 \@safe@activestruel\org@pageref{#1}\@safe@activesfalse}
3810 \fi
3811 \else
3812 \let\org@ref\ref
3813 \let\org@pageref\pageref
3814 \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.

```

3815 \bbl@xin@{B}\bbl@opt@safe
3816 \ifin@
3817 \bbl@redefine\@citex[#1]#2{%
3818 \@safe@activestruel\edef\@tempa{#2}\@safe@activesfalse
3819 \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.

```

3820 \AtBeginDocument{%
3821 \@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.)

```

3822 \def\@citex[#1][#2]#3{%
3823 \@safe@activestruel\edef\@tempa{#3}\@safe@activesfalse
3824 \org@@citex[#1][#2]{\@tempa}}%
3825 }{}

```

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

```

3826 \AtBeginDocument{%
3827 \@ifpackageloaded{cite}{%
3828 \def\@citex[#1]#2{%
3829 \@safe@activestruel\org@@citex[#1][#2]\@safe@activesfalse}%
3830 }{}

```

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

```

3831 \bbl@redefine\nocite#1{%
3832 \@safe@activestruel\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@activestru` 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.

```
3833 \bbl@redefine\bibcite{%
3834   \bbl@cite@choice
3835   \bibcite}
```

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

```
3836 \def\bbl@bibcite#1#2{%
3837   \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.

```
3838 \def\bbl@cite@choice{%
3839   \global\let\bibcite\bbl@bibcite
3840   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
3841   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
3842   \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.

```
3843 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3844 \bbl@redefine\@bibitem#1{%
3845   \@safe@activestruer\org@bibitem{#1}\@safe@activesfalse}
3846 \else
3847   \let\org@nocite\nocite
3848   \let\org@citex\@citex
3849   \let\org@bibcite\bibcite
3850   \let\org@bibitem\@bibitem
3851 \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.

```
3852 \bbl@trace{Marks}
3853 \IfBabelLayout{sectioning}
3854   {\ifx\bbl@opt@headfoot\@nnil
3855     \g@addto@macro\@resetactivechars{%
3856       \set@typeset@protect
3857       \expandafter\select@language\x\expandafter{\bbl@main@language}%
3858       \let\protect\noexpand
3859       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3860         \edef\thepage{%
3861           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3862       \fi}%
3863   \fi}
3864 {\ifbbl@single\else
3865   \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
3866   \markright#1{%
3867     \bbl@ifblank{#1}%
```

```

3868     {\org@markright{}}%
3869     {\toks@#1}%
3870     \bbl@exp{%
3871         \\org@markright{\\protect\\foreignlanguage{\language}%
3872             {\\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.)

```

3873     \ifx\@mkboth\markboth
3874         \def\bbl@tempc{\let\@mkboth\markboth}%
3875     \else
3876         \def\bbl@tempc{%
3877             \fi
3878             \bbl@ifunset{markboth }{\bbl@redefine\bbl@redefineroobust
3879             \markboth#1#2{%
3880                 \protected@edef\bbl@tempb##1{%
3881                     \protect\foreignlanguage
3882                     {\language}{\protect\bbl@restore@actives##1}}%
3883                 \bbl@ifblank{#1}%
3884                 {\toks@{}}%
3885                 {\toks@\expandafter{\bbl@tempb{#1}}}%
3886                 \bbl@ifblank{#2}%
3887                 {\@temptokena{}}%
3888                 {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3889                 \bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}}%
3890                 \bbl@tempc
3891             \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.

```

3892 \bbl@trace{Preventing clashes with other packages}
3893 \ifx\org@ref\undefined\else
3894     \bbl@xin@{R}\bbl@opt@safe
3895     \ifin@
3896         \AtBeginDocument{%
3897             \@ifpackageloaded{ifthen}{%
3898                 \bbl@redefine@long\ifthenelse#1#2#3{%
3899                     \let\bbl@temp@pref\pageref
3900                     \let\pageref\org@pageref
3901                     \let\bbl@temp@ref\ref
3902                     \let\ref\org@ref

```

```

3903     \@safe@activestru
3904     \org@ifthenelse{#1}%
3905     {\let\pageref\bbl@temp@pref
3906     \let\ref\bbl@temp@ref
3907     \@safe@activesfalse
3908     #2}%
3909     {\let\pageref\bbl@temp@pref
3910     \let\ref\bbl@temp@ref
3911     \@safe@activesfalse
3912     #3}%
3913     }%
3914   }{}%
3915 }
3916 \fi

```

5.3.2 varioref

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

```

3917 \AtBeginDocument{%
3918   \@ifpackageloaded{varioref}{%
3919     \bbl@redefine\@@vpageref#1[#2]#3{%
3920       \@safe@activestru
3921       \org@@@vpageref{#1}[#2]{#3}%
3922       \@safe@activesfalse}%
3923     \bbl@redefine\vrefpagemum#1#2{%
3924       \@safe@activestru
3925       \org@vrefpagemum{#1}{#2}%
3926       \@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.

```

3927   \expandafter\def\csname Ref \endcsname#1{%
3928     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3929   }{}%
3930 }
3931 \fi

```

5.3.3 hpline

`\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.

```

3932 \AtEndOfPackage{%
3933   \AtBeginDocument{%
3934     \@ifpackageloaded{hhline}%
3935     {\expandafter\ifx\csname normal@char\string\endcsname\relax
3936     \else
3937       \makeatletter
3938       \def\@currname{hhline}\input{hhline.sty}\makeatother
3939       \fi}%
3940     {}}

```

`\substitutefontfamily` *Deprecated*. Use the tools provided by \LaTeX . 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.

```

3941 \def\substitutefontfamily#1#2#3{%
3942   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3943   \immediate\write15{%
3944     \string\ProvidesFile{#1#2.fd}%
3945     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3946     \space generated font description file]^J
3947     \string\DeclareFontFamily{#1}{#2}{}^^J
3948     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
3949     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
3950     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
3951     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
3952     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
3953     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
3954     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
3955     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
3956   }%
3957   \closeout15
3958 }
3959 \@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`

```

3960 \bbl@trace{Encoding and fonts}
3961 \newcommand\BabelNonASCII{LGR,LGI,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3962 \newcommand\BabelNonText{TS1,T3,TS3}
3963 \let\org@TeX\TeX
3964 \let\org@LaTeX\LaTeX
3965 \let\ensureascii@firstofone
3966 \let\asciientcoding@empty
3967 \AtBeginDocument{%
3968   \def\@elt#1{,#1,}%
3969   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3970   \let\@elt\relax
3971   \let\bbl@tempb@empty
3972   \def\bbl@tempc{OT1}%
3973   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3974     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3975   \bbl@foreach\bbl@tempa{%
3976     \bbl@xin@{,#1,}{,\BabelNonASCII,}%
3977     \ifin@
3978       \def\bbl@tempb{#1}% Store last non-ascii
3979     \else\bbl@xin@{,#1,}{,\BabelNonText,}% Pass
3980     \ifin@else
3981       \def\bbl@tempc{#1}% Store last ascii
3982     \fi
3983   \fi}%
3984   \ifx\bbl@tempb@empty\else
3985     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3986     \ifin@else
3987       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3988     \fi
3989     \let\asciientcoding\bbl@tempc
3990     \renewcommand\ensureascii[1]{%
3991       {\fontencoding{\asciientcoding}\selectfont#1}}%
3992     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3993     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%

```

```
3994 \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.

```
3995 \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.

```
3996 \AtBeginDocument{%
3997   \ifpackageloaded{fontspec}%
3998     {\xdef\latinencoding{%
3999       \ifx\UTFencname\undefined
4000         EU\ifcase\bb@engine\or2\or1\fi
4001       \else
4002         \UTFencname
4003       \fi}}%
4004   {\gdef\latinencoding{OT1}%
4005     \ifx\cf@encoding\bb@t@one
4006       \xdef\latinencoding{\bb@t@one}%
4007     \else
4008       \def\@elt#1{,#1,%
4009         \edef\bb@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
4010         \let\@elt\relax
4011         \bb@xin@{,T1,}\bb@tempa
4012         \ifin@
4013           \xdef\latinencoding{\bb@t@one}%
4014         \fi
4015       \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.

```
4016 \DeclareRobustCommand{\latintext}{%
4017   \fontencoding{\latinencoding}\selectfont
4018   \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.

```
4019 \ifx\undefined\DeclareTextFontCommand
4020   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
4021 \else
4022   \DeclareTextFontCommand{\textlatin}{\latintext}
4023 \fi
```

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

```
4024 \def\bb@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 `LuaTEX-j` shows, vertical typesetting is possible, too.

```

4025 \bbl@trace{Loading basic (internal) bidi support}
4026 \ifodd\bbl@engine
4027 \else % TODO. Move to txtbabel
4028   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200 % Any xe+lua bidi=
4029     \bbl@error
4030     {The bidi method 'basic' is available only in\\%
4031     luatex. I'll continue with 'bidi=default', so\\%
4032     expect wrong results}%
4033     {See the manual for further details.}%
4034     \let\bbl@beforeforeign\leavevmode
4035     \AtEndOfPackage{%
4036       \EnableBabelHook{babel-bidi}%
4037       \bbl@xebidipar}
4038   \fi\fi
4039   \def\bbl@loadxebidi#1{%
4040     \ifx\RTLfootnotetext\undefined
4041       \AtEndOfPackage{%
4042         \EnableBabelHook{babel-bidi}%
4043         \bbl@loadfontspec % bidi needs fontspec
4044         \usepackage#1{bidi}%
4045         \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
4046         \def\DigitsDotDashInterCharToks{% See the 'bidi' package
4047           \ifnum\@nameuse{bbl@wdir@\languagename}=\tw@ % 'AL' bidi
4048             \bbl@digitsdotdash % So ignore in 'R' bidi
4049           \fi}}%
4050     \fi}
4051   \ifnum\bbl@bidimode>200 % Any xe bidi=
4052     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
4053       \bbl@tentative{bidi=bidi}
4054       \bbl@loadxebidi{}
4055     \or
4056       \bbl@loadxebidi{[rldocument]}
4057     \or
4058       \bbl@loadxebidi{}
4059     \fi
4060   \fi
4061 \fi
4062 % TODO? Separate:
4063 \ifnum\bbl@bidimode=\@ne % Any bidi= except default=1
4064   \let\bbl@beforeforeign\leavevmode
4065   \ifodd\bbl@engine
4066     \newattribute\bbl@attr@dir
4067     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
4068     \bbl@exp{\output{\bodydir\pagedir\the\output}}
4069   \fi
4070   \AtEndOfPackage{%
4071     \EnableBabelHook{babel-bidi}%
4072     \ifodd\bbl@engine\else
4073       \bbl@xebidipar
4074     \fi}

```


4075 \fi

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

```
4076 \bbl@trace{Macros to switch the text direction}
4077 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
4078 \def\bbl@rscripts{% TODO. Base on codes ??
4079   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
4080   Old Hungarian,Lydian,Mandaean,Manichaeen,%
4081   Meroitic Cursive,Meroitic,Old North Arabian,%
4082   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
4083   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
4084   Old South Arabian,}%
4085 \def\bbl@provide@dirs#1{%
4086   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
4087   \ifin@
4088     \global\bbl@csarg\chardef{wdir@#1}\@ne
4089     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4090     \ifin@
4091     \global\bbl@csarg\chardef{wdir@#1}\tw@
4092     \fi
4093   \else
4094     \global\bbl@csarg\chardef{wdir@#1}\z@
4095     \fi
4096   \ifodd\bbl@engine
4097     \bbl@csarg\ifcase{wdir@#1}%
4098     \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
4099     \or
4100     \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
4101     \or
4102     \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
4103     \fi
4104   \fi}
4105 \def\bbl@switchdir{%
4106   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4107   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
4108   \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}%
4109 \def\bbl@setdirs#1{% TODO - math
4110 \ifcase\bbl@select@type % TODO - strictly, not the right test
4111   \bbl@bodydir{#1}%
4112   \bbl@pardir{#1}% <- Must precede \bbl@textdir
4113   \fi
4114   \bbl@textdir{#1}}
4115 % TODO. Only if \bbl@bidimode > 0?:
4116 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4117 \DisableBabelHook{babel-bidi}
```

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

```
4118 \ifodd\bbl@engine % luatex=1
4119 \else % pdftex=0, xetex=2
4120   \newcount\bbl@dirlevel
4121   \chardef\bbl@thetextdir\z@
4122   \chardef\bbl@thepardir\z@
4123   \def\bbl@textdir#1{%
4124     \ifcase#1\relax
4125       \chardef\bbl@thetextdir\z@
4126       \@nameuse{setlatin}%
4127       \bbl@textdir@i\beginL\endL
4128     \else
4129       \chardef\bbl@thetextdir\@ne
4130       \@nameuse{setnonlatin}%
4131       \bbl@textdir@i\beginR\endR
4132     \fi}
4133   \def\bbl@textdir@i#1#2{%
```

```

4134 \ifhmode
4135   \ifnum\currentgrouplevel>\z@
4136     \ifnum\currentgrouplevel=\bbl@dirlevel
4137       \bbl@error{Multiple bidi settings inside a group}%
4138       {I'll insert a new group, but expect wrong results.}%
4139       \bgroup\aftergroup#2\aftergroup\egroup
4140     \else
4141       \ifcase\currentgrouptype\or % 0 bottom
4142         \aftergroup#2% 1 simple {}
4143       \or
4144         \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4145       \or
4146         \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4147       \or\or\or % vbox vtop align
4148       \or
4149         \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4150       \or\or\or\or\or\or % output math disc insert vcent mathchoice
4151       \or
4152         \aftergroup#2% 14 \beginngroup
4153       \else
4154         \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4155       \fi
4156     \fi
4157     \bbl@dirlevel\currentgrouplevel
4158   \fi
4159   #1%
4160 \fi}
4161 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4162 \let\bbl@bodydir\@gobble
4163 \let\bbl@pagedir\@gobble
4164 \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).

```

4165 \def\bbl@xebidipar{%
4166   \let\bbl@xebidipar\relax
4167   \TeXeTstate\@ne
4168   \def\bbl@xeeverypar{%
4169     \ifcase\bbl@thepardir
4170       \ifcase\bbl@thetextdir\else\beginR\fi
4171     \else
4172       {\setbox\z@\lastbox\beginR\box\z@}%
4173     \fi}%
4174   \let\bbl@severypar\everypar
4175   \newtoks\everypar
4176   \everypar=\bbl@severypar
4177   \bbl@severypar{\bbl@xeeverypar\the\everypar}}
4178 \ifnum\bbl@bidimode>200 % Any xe bidi=
4179   \let\bbl@textdiri\@gobbletwo
4180   \let\bbl@xebidipar\@empty
4181   \AddBabelHook{bidi}{foreign}{%
4182     \def\bbl@tempa{\def\BabelText###1}%
4183     \ifcase\bbl@thetextdir
4184       \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4185     \else
4186       \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4187     \fi}
4188   \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4189 \fi
4190 \fi

```

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

```

4191 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}

```

```

4192 \AtBeginDocument{%
4193   \ifx\pdfstringdefDisableCommands\@undefined\else
4194     \ifx\pdfstringdefDisableCommands\relax\else
4195       \pdfstringdefDisableCommands{\let\babelsubl\@firstofone}%
4196     \fi
4197   \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`.

```

4198 \bbl@trace{Local Language Configuration}
4199 \ifx\loadlocalcfg\@undefined
4200   \@ifpackagewith{babel}{noconfigs}%
4201   {\let\loadlocalcfg\@gobble}%
4202   {\def\loadlocalcfg#1{%
4203     \InputIfFileExists{#1.cfg}%
4204     {\typeout{*****^J%
4205              * Local config file #1.cfg used^J%
4206              *}}%
4207     \@empty}}
4208 \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).

```

4209 \bbl@trace{Language options}
4210 \let\bbl@afterlang\relax
4211 \let\BabelModifiers\relax
4212 \let\bbl@loaded\@empty
4213 \def\bbl@load@language#1{%
4214   \InputIfFileExists{#1.ldf}%
4215   {\edef\bbl@loaded{\CurrentOption
4216     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4217     \expandafter\let\expandafter\bbl@afterlang
4218       \csname\CurrentOption.ldf-h@k\endcsname
4219     \expandafter\let\expandafter\BabelModifiers
4220       \csname bbl@mod@\CurrentOption\endcsname
4221     \bbl@exp{\AtBeginDocument{%
4222       \bbl@usehooks@lang{\CurrentOption}{\begin{document}}{\CurrentOption}}}%
4223     {\bbl@error{%
4224       Unknown option '\CurrentOption'. Either you misspelled it\%
4225       or the language definition file \CurrentOption.ldf was not found}%
4226       Valid options are, among others: shorthands=, KeepShorthandsActive,\%
4227       activeacute, activegrave, noconfigs, safe=, main=, math=\%
4228       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.

```

4229 \def\bbl@try@load@lang#1#2#3{%
4230   \IfFileExists{\CurrentOption.ldf}%
4231   {\bbl@load@language{\CurrentOption}}%
4232   {#1\bbl@load@language{#2}#3}}
4233 %
4234 \DeclareOption{hebrew}{%
4235   \input{rlbabel.def}%

```

```

4236 \bbl@load@language{hebrew}}
4237 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4238 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4239 \DeclareOption{northernsami}{\bbl@try@load@lang{}{samin}{}}
4240 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4241 \DeclareOption{polutonikogreek}{%
4242 \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4243 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4244 \DeclareOption{scottishgaelic}{\bbl@try@load@lang{}{scottish}{}}
4245 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4246 \DeclareOption{upporsorbian}{\bbl@try@load@lang{}{usorbian}{}}

```

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

```

4247 \ifx\bbl@opt@config@nnil
4248 \@ifpackagewith{babel}{noconfigs}{}%
4249   {\InputIfFileExists{bblopts.cfg}%
4250     {\typeout{*****^J%
4251               * Local config file bblopts.cfg used^^J%
4252               *}}}%
4253   {}}%
4254 \else
4255 \InputIfFileExists{\bbl@opt@config.cfg}%
4256   {\typeout{*****^J%
4257             * Local config file \bbl@opt@config.cfg used^^J%
4258             *}}%
4259   {\bbl@error{%
4260     Local config file '\bbl@opt@config.cfg' not found}{%
4261     Perhaps you misspelled it.}}%
4262 \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.

```

4263 \ifx\bbl@opt@main@nnil
4264 \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4265   \let\bbl@tempb@empty
4266   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4267   \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4268   \bbl@foreach\bbl@tempb{% \bbl@tempb is a reversed list
4269     \ifx\bbl@opt@main@nnil % ie, if not yet assigned
4270       \ifodd\bbl@iniflag % = *=
4271         \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}}%
4272     \else % n +=
4273       \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}}%
4274     \fi
4275   \fi}%
4276 \fi
4277 \else
4278 \bbl@info{Main language set with 'main='. Except if you have\\%
4279           problems, prefer the default mechanism for setting\\%
4280           the main language, ie, as the last declared.\\%
4281           Reported}
4282 \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`).

```

4283 \ifx\bbl@opt@main@nnil\else

```

```

4284 \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
4285 \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4286 \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.

```

4287 \bbl@foreach\bbl@language@opts{%
4288 \def\bbl@tempa{#1}%
4289 \ifx\bbl@tempa\bbl@opt@main\else
4290 \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4291 \bbl@ifunset{ds@#1}%
4292 {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4293 }%
4294 \else % + * (other = ini)
4295 \DeclareOption{#1}{%
4296 \bbl@ldfinit
4297 \babelprovide[import]{#1}%
4298 \bbl@afterldf{}}%
4299 \fi
4300 \fi}
4301 \bbl@foreach\@classoptionslist{%
4302 \def\bbl@tempa{#1}%
4303 \ifx\bbl@tempa\bbl@opt@main\else
4304 \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4305 \bbl@ifunset{ds@#1}%
4306 {\IfFileExists{#1. ldf}%
4307 {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4308 }%
4309 }%
4310 \else % + * (other = ini)
4311 \IfFileExists{babel-#1.tex}%
4312 {\DeclareOption{#1}{%
4313 \bbl@ldfinit
4314 \babelprovide[import]{#1}%
4315 \bbl@afterldf{}}}%
4316 }%
4317 \fi
4318 \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):

```

4319 \def\AfterBabelLanguage#1{%
4320 \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang{}}
4321 \DeclareOption*{}
4322 \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.

```

4323 \bbl@trace{Option 'main'}
4324 \ifx\bbl@opt@main@nnil
4325 \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4326 \let\bbl@tempc\@empty
4327 \edef\bbl@templ{,\bbl@loaded,}
4328 \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4329 \bbl@for\bbl@tempb\bbl@tempa{%
4330 \edef\bbl@tempd{,\bbl@tempb,}%
4331 \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4332 \bbl@xin{\bbl@tempd}{\bbl@templ}%

```

```

4333 \ifin@edef\bbl@tempc{\bbl@tempb}\fi}
4334 \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4335 \expandafter\bbl@tempa\bbl@loaded,\@nnil
4336 \ifx\bbl@tempb\bbl@tempc\else
4337 \bbl@warning{%
4338 Last declared language option is '\bbl@tempc',\%
4339 but the last processed one was '\bbl@tempb'.\%
4340 The main language can't be set as both a global\%
4341 and a package option. Use 'main=\bbl@tempc' as\%
4342 option. Reported}
4343 \fi
4344 \else
4345 \ifodd\bbl@iniflag % case 1,3 (main is ini)
4346 \bbl@ldfinit
4347 \let\CurrentOption\bbl@opt@main
4348 \bbl@exp{% \bbl@opt@provide = empty if *
4349 \\\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4350 \bbl@afterldf{}
4351 \DeclareOption{\bbl@opt@main}{}
4352 \else % case 0,2 (main is ldf)
4353 \ifx\bbl@loadmain\relax
4354 \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4355 \else
4356 \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4357 \fi
4358 \ExecuteOptions{\bbl@opt@main}
4359 \@namedef{ds@\bbl@opt@main}{}%
4360 \fi
4361 \DeclareOption*{}
4362 \ProcessOptions*
4363 \fi
4364 \bbl@exp{%
4365 \\\AtBeginDocument{\\\bbl@usehooks@lang{/}{\beginDocument}{\}}}%
4366 \def\AfterBabelLanguage{%
4367 \bbl@error
4368 {Too late for \string\AfterBabelLanguage}%
4369 {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.

4370 \ifx\bbl@main@language\@undefined
4371 \bbl@info{%
4372 You haven't specified a language as a class or package\%
4373 option. I'll load 'nil'. Reported}
4374 \bbl@load@language{nil}
4375 \fi
4376 \</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

```
4377 \<*kernel>
```

```

4378 \let\bbl@onlyswitch\@empty
4379 \input babel.def
4380 \let\bbl@onlyswitch\@undefined
4381 </kernel>
4382 <*patterns>

```

7 Loading hyphenation patterns

The following code is meant to be read by $\text{\texttt{iniT\TeX}}$ because it should instruct $\text{\texttt{T\TeX}}$ 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.

```

4383 <<Make sure ProvidesFile is defined>>
4384 \ProvidesFile{hyphen.cfg}[<<date>> v<<version>> Babel hyphens]
4385 \xdef\bbl@format{\jobname}
4386 \def\bbl@version{<<version>>}
4387 \def\bbl@date{<<date>>}
4388 \ifx\AtBeginDocument\@undefined
4389   \def\@empty{}
4390 \fi
4391 <<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.

```

4392 \def\process@line#1#2 #3 #4 {%
4393   \ifx=#1%
4394     \process@synonym{#2}%
4395   \else
4396     \process@language{#1#2}{#3}{#4}%
4397   \fi
4398   \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.

```

4399 \toks@{}
4400 \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.

```

4401 \def\process@synonym#1{%
4402   \ifnum\last@language=\m@ne
4403     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4404   \else
4405     \expandafter\chardef\csname l@#1\endcsname\last@language
4406     \wlog{\string\l@#1=\string\language\the\last@language}%
4407     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4408       \csname\languagenamename hyphenmins\endcsname
4409     \let\bbl@elt\relax
4410     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}{}%
4411   \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 `\<lang>hyphenmins` macro. When no assignments were made we provide a default setting.

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

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

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

`\bbl@elt{<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.

```

4412 \def\process@language#1#2#3{%
4413   \expandafter\addlanguage\csname l@#1\endcsname
4414   \expandafter\language\csname l@#1\endcsname
4415   \edef\languagename{#1}%
4416   \bbl@hook@everylanguage{#1}%
4417   % > luatex
4418   \bbl@get@enc#1: : \@@@
4419   \begingroup
4420     \lefthyphenmin@m@ne
4421     \bbl@hook@loadpatterns{#2}%
4422     % > luatex
4423     \ifnum\lefthyphenmin=m@ne
4424     \else
4425       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4426         \the\lefthyphenmin\the\righthyphenmin}%
4427     \fi
4428   \endgroup
4429   \def\bbl@tempa{#3}%
4430   \ifx\bbl@tempa@empty\else
4431     \bbl@hook@loadexceptions{#3}%
4432     % > luatex
4433   \fi
4434   \let\bbl@elt\relax
4435   \edef\bbl@languages{%
4436     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4437   \ifnum\the\language=z@
4438     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4439       \set@hyphenmins\tw@\thr@\relax
4440     \else
4441       \expandafter\expandafter\expandafter\set@hyphenmins
4442       \csname #1hyphenmins\endcsname
4443     \fi
4444     \the\toks@
4445     \toks@{}%
4446   \fi}

```

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

```
4447 \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.


```

4448 \def\bb@hook@everylanguage#1{
4449 \def\bb@hook@loadpatterns#1{\input #1\relax}
4450 \let\bb@hook@loadexceptions\bb@hook@loadpatterns
4451 \def\bb@hook@loadkernel#1{%
4452 \def\addlanguage{\csname newlanguage\endcsname}%
4453 \def\adddialect##1##2{%
4454 \global\chardef##1##2\relax
4455 \wlog{\string##1 = a dialect from \string\language##2}}%
4456 \def\iflanguage##1{%
4457 \expandafter\ifx\csname l@##1\endcsname\relax
4458 \@nolanerr{##1}%
4459 \else
4460 \ifnum\csname l@##1\endcsname=\language
4461 \expandafter\expandafter\expandafter\@firstoftwo
4462 \else
4463 \expandafter\expandafter\expandafter\@secondoftwo
4464 \fi
4465 \fi}%
4466 \def\providehyphenmins##1##2{%
4467 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4468 \@namedef{##1hyphenmins}{##2}%
4469 \fi}%
4470 \def\set@hyphenmins##1##2{%
4471 \lefthyphenmin##1\relax
4472 \righthyphenmin##2\relax}%
4473 \def\selectlanguage{%
4474 \errhelp{Selecting a language requires a package supporting it}%
4475 \errmessage{Not loaded}}%
4476 \let\foreignlanguage\selectlanguage
4477 \let\otherlanguage\selectlanguage
4478 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4479 \def\bb@usehooks##1##2{% TODO. Temporary!!
4480 \def\setlocale{%
4481 \errhelp{Find an armchair, sit down and wait}%
4482 \errmessage{Not yet available}}%
4483 \let\uselocale\setlocale
4484 \let\locale\setlocale
4485 \let\selectlocale\setlocale
4486 \let\localename\setlocale
4487 \let\textlocale\setlocale
4488 \let\textlanguage\setlocale
4489 \let\languagetext\setlocale}
4490 \begingroup
4491 \def\AddBabelHook#1#2{%
4492 \expandafter\ifx\csname bb@hook@#2\endcsname\relax
4493 \def\next{\toks1}%
4494 \else
4495 \def\next{\expandafter\gdef\csname bb@hook@#2\endcsname###1}%
4496 \fi
4497 \next}
4498 \ifx\directlua@undefined
4499 \ifx\XeTeXinputencoding@undefined\else
4500 \input xebabel.def
4501 \fi
4502 \else
4503 \input luababel.def
4504 \fi
4505 \openin1 = babel-\bb@format.cfg
4506 \ifeof1
4507 \else
4508 \input babel-\bb@format.cfg\relax
4509 \fi
4510 \closein1

```

```
4511 \endgroup
4512 \bbl@hook@loadkernel{switch.def}
```

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

```
4513 \openin1 = language.dat
```

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

```
4514 \def\languagename{english}%
4515 \ifeof1
4516 \message{I couldn't find the file language.dat,\space
4517         I will try the file hyphen.tex}
4518 \input hyphen.tex\relax
4519 \chardef\l@english\z@
4520 \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`.

```
4521 \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.

```
4522 \loop
4523   \endlinechar\m@ne
4524   \read1 to \bbl@line
4525   \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.

```
4526   \if T\ifeof1\fi T\relax
4527   \ifx\bbl@line@empty\else
4528     \edef\bbl@line{\bbl@line\space\space\space}%
4529     \expandafter\process@line\bbl@line\relax
4530   \fi
4531 \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.

```
4532 \begingroup
4533   \def\bbl@elt#1#2#3#4{%
4534     \global\language=#2\relax
4535     \gdef\languagename{#1}%
4536     \def\bbl@elt##1##2##3##4{}}%
4537   \bbl@languages
4538 \endgroup
4539 \fi
4540 \closein1
```

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

```
4541 \if/\the\toks@/\else
4542   \errhelp{language.dat loads no language, only synonyms}
4543   \errmessage{Orphan language synonym}
4544 \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.

```
4545 \let\bbl@line@undefined
4546 \let\process@line@undefined
4547 \let\process@synonym@undefined
```

```

4548 \let\process@language\@undefined
4549 \let\bbl@get@enc\@undefined
4550 \let\bbl@hyph@enc\@undefined
4551 \let\bbl@tempa\@undefined
4552 \let\bbl@hook@loadkernel\@undefined
4553 \let\bbl@hook@everylanguage\@undefined
4554 \let\bbl@hook@loadpatterns\@undefined
4555 \let\bbl@hook@loadexceptions\@undefined
4556 \</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].

```

4557 <<{*More package options}>> ≡
4558 \chardef\bbl@bidimode\z@
4559 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4560 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4561 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4562 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4563 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4564 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4565 <</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.

```

4566 <<{*Font selection}>> ≡
4567 \bbl@trace{Font handling with fontspec}
4568 \ifx\ExplSyntax0n\@undefined\else
4569   \def\bbl@fs@warn@nx#1#2{% \bbl@tempfs is the original macro
4570     \in@{,#1,}{,no-script,language-not-exist,}%
4571     \ifin@else\bbl@tempfs@nx{#1}{#2}\fi}
4572   \def\bbl@fs@warn@nxx#1#2#3{%
4573     \in@{,#1,}{,no-script,language-not-exist,}%
4574     \ifin@else\bbl@tempfs@nxx{#1}{#2}{#3}\fi}
4575   \def\bbl@loadfontspec{%
4576     \let\bbl@loadfontspec\relax
4577     \ifx\fontspec\@undefined
4578       \usepackage{fontspec}%
4579     \fi}%
4580 \fi
4581 \@onlypreamble\babelfont
4582 \newcommand\babelfont[2][]{% 1=langs/scripts 2=fam
4583   \bbl@foreach{#1}{%
4584     \expandafter\ifx\cscname date##1\endcscname\relax
4585     \IfFileExists{babel-##1.tex}%
4586       {\babelprovide{##1}}%
4587     }%
4588   \fi}%
4589 \edef\bbl@tempa{#1}%
4590 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4591 \bbl@loadfontspec
4592 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4593 \bbl@bblfont}
4594 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
4595   \bbl@ifunset{\bbl@tempb family}%
4596   {\bbl@providefam{\bbl@tempb}}%

```

```

4597 {}%
4598 % For the default font, just in case:
4599 \bbl@ifunset{\bbl@lsys@\languagename}\bbl@provide@lsys{\languagename}}{}%
4600 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4601 {\bbl@csarg\edef{\bbl@tempb dflt@}<{#1}{#2}}% save bbl@rmdflt@
4602 \bbl@exp{%
4603 \let<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
4604 \\bbl@font@set<bbl@\bbl@tempb dflt@\languagename>%
4605 \<bbl@tempb default>\<bbl@tempb family>}}%
4606 {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4607 \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:

```

4608 \def\bbl@providefam#1{%
4609 \bbl@exp{%
4610 \\newcommand<#1default>{}% Just define it
4611 \\bbl@add@list\\bbl@font@fams{#1}%
4612 \\DeclareRobustCommand<#1family>{%
4613 \not@math@alphabet<#1family>\relax
4614 % \\prepare@family@series@update{#1}<#1default>% TODO. Fails
4615 \\fontfamily<#1default>%
4616 \<ifx>\\UseHooks\\<undefined>\<else>\\UseHook{#1family}\<fi>%
4617 \\selectfont}%
4618 \\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.

```

4619 \def\bbl@nostdfont#1{%
4620 \bbl@ifunset{\bbl@WFF@f@family}%
4621 {\bbl@csarg\gdef{\bbl@WFF@f@family}}% Flag, to avoid dupl warns
4622 \bbl@infowarn{The current font is not a babel standard family:\\%
4623 #1%
4624 \fontname\font\\%
4625 There is nothing intrinsically wrong with this warning, and\\%
4626 you can ignore it altogether if you do not need these\\%
4627 families. But if they are used in the document, you should be\\%
4628 aware 'babel' will not set Script and Language for them, so\\%
4629 you may consider defining a new family with \string\babelfont.\\%
4630 See the manual for further details about \string\babelfont.\\%
4631 Reported}}
4632 {}}%
4633 \gdef\bbl@switchfont{%
4634 \bbl@ifunset{\bbl@lsys@\languagename}\bbl@provide@lsys{\languagename}}{}%
4635 \bbl@exp{% eg Arabic -> arabic
4636 \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}%
4637 \bbl@foreach\bbl@font@fams{%
4638 \bbl@ifunset{\bbl@##1dflt@\languagename}% (1) language?
4639 {\bbl@ifunset{\bbl@##1dflt*@\bbl@tempa}% (2) from script?
4640 {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
4641 {}% 123=F - nothing!
4642 {\bbl@exp{% 3=T - from generic
4643 \global\let<bbl@##1dflt@\languagename>%
4644 \<bbl@##1dflt@>}}}%
4645 {\bbl@exp{% 2=T - from script
4646 \global\let<bbl@##1dflt@\languagename>%
4647 \<bbl@##1dflt*@\bbl@tempa>}}}%
4648 {}}% 1=T - language, already defined
4649 \def\bbl@tempa{\bbl@nostdfont}}% TODO. Don't use \bbl@tempa
4650 \bbl@foreach\bbl@font@fams{% don't gather with prev for
4651 \bbl@ifunset{\bbl@##1dflt@\languagename}%
4652 {\bbl@cs{famrst@##1}%
4653 \global\bbl@csarg\let{famrst@##1}\relax}%
4654 {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4655 \\bbl@add\\originalTeX{%

```

```

4656         \\\bbl@font@rst{\bbl@cl{##ldflt}}%
4657             \<##ldefault>\<##lfamily>{##1}}%
4658         \\\bbl@font@set<\bbl@##ldflt@<language>% the main part!
4659             \<##ldefault>\<##lfamily>}}}%
4660     \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`.

```

4661 \ifx\@family\undefined\else % if latex
4662 \ifcase\bbl@engine % if pdftex
4663 \let\bbl@cckcstdfonts\relax
4664 \else
4665 \def\bbl@cckcstdfonts{%
4666 \begingroup
4667 \global\let\bbl@cckcstdfonts\relax
4668 \let\bbl@tempa\empty
4669 \bbl@foreach\bbl@font@fams{%
4670 \bbl@ifunset{\bbl@##ldflt@}%
4671 {\@nameuse{##lfamily}}%
4672 \bbl@csarg\gdef{WFF@f@family}}% Flag
4673 \bbl@exp{\bbl@add{\bbl@tempa{* \<##lfamily>= \f@family\\}%
4674 \space\space\fontname\font\\}}%
4675 \bbl@csarg\xdef{##ldflt@}{\f@family}%
4676 \expandafter\xdef\csname ##ldefault\endcsname{\f@family}}%
4677 {}}%
4678 \ifx\bbl@tempa\empty\else
4679 \bbl@infowarn{The following font families will use the default\\%
4680 settings for all or some languages:\\%
4681 \bbl@tempa
4682 There is nothing intrinsically wrong with it, but\\%
4683 'babel' will no set Script and Language, which could\\%
4684 be relevant in some languages. If your document uses\\%
4685 these families, consider redefining them with \string\babelfont.\\%
4686 Reported}%
4687 \fi
4688 \endgroup}
4689 \fi
4690 \fi

```

Now the macros defining the font with `fontspec`.

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

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

```

4691 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4692 \bbl@xin@{<>}{#1}%
4693 \ifin@
4694 \bbl@exp{\bbl@fontspec@set\#1\expandafter@gobbletwo\#1\#3}%
4695 \fi
4696 \bbl@exp{%
4697 \def\#2{#1}% eg, \rmdefault{\bbl@rmdflt@lang}
4698 \bbl@ifsamestring{#2}{\f@family}%
4699 {\#3%
4700 \bbl@ifsamestring{\f@series}{\bfdefault}{\bfseries}}%
4701 \let\bbl@tempa\relax}%
4702 {}}
4703 % TODO - next should be global?, but even local does its job. I'm
4704 % still not sure -- must investigate:
4705 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily

```

```

4706 \let\bbl@tempe\bbl@mapselect
4707 \edef\bbl@tempb{\bbl@stripslash#4/}% Catcodes hack (better pass it).
4708 \bbl@exp{\bbl@replace\bbl@tempb{\bbl@stripslash\family/}}}%
4709 \let\bbl@mapselect\relax
4710 \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4711 \let#4\@empty % Make sure \renewfontfamily is valid
4712 \bbl@exp{%
4713 \let\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
4714 <\keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4715 {\bbl@cl{fontspec-script}{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4716 <\keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4717 {\bbl@cl{fontspec-language}{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4718 \let\bbl@temp@fs@nx<__fontspec_warning:nx>%
4719 \let<__fontspec_warning:nx>\bbl@fs@warn@nx
4720 \let\bbl@temp@fs@nxx<__fontspec_warning:nxx>%
4721 \let<__fontspec_warning:nxx>\bbl@fs@warn@nxx
4722 \renewfontfamily\#4%
4723 [\bbl@cl{lsys},%
4724 \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
4725 #2}{#3}% ie \bbl@exp{..}{#3}
4726 \bbl@exp{%
4727 \let<__fontspec_warning:nx>\bbl@temp@fs@nx
4728 \let<__fontspec_warning:nxx>\bbl@temp@fs@nxx}%
4729 \beginngroup
4730 #4%
4731 \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4732 \endgroup % TODO. Find better tests:
4733 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4734 {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
4735 \ifin@
4736 \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
4737 \fi
4738 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4739 {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
4740 \ifin@
4741 \global\bbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
4742 \fi
4743 \let#4\bbl@temp@fam
4744 \bbl@exp{\let<\bbl@stripslash#4\space>\bbl@temp@pfam
4745 \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.

```

4746 \def\bbl@font@rst#1#2#3#4{%
4747 \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.

```

4748 \def\bbl@font@fams{rm,sf,tt}
4749 <</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.

```

4750 <<(*Footnote changes)>> ≡
4751 \bbl@trace{Bidi footnotes}
4752 \ifnum\bbl@bidimode>\z@ % Any bidi=
4753 \def\bbl@footnote#1#2#3{%
4754 \ifnextchar[%
4755 {\bbl@footnote@o{#1}{#2}{#3}}%

```

```

4756     {\bbl@footnote@x{#1}{#2}{#3}}
4757 \long\def\bbl@footnote@x#1#2#3#4{%
4758   \bgroup
4759     \select@language@x{\bbl@main@language}%
4760     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4761   \egroup}
4762 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4763   \bgroup
4764     \select@language@x{\bbl@main@language}%
4765     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4766   \egroup}
4767 \def\bbl@footnotetext#1#2#3{%
4768   \@ifnextchar[%
4769     {\bbl@footnotetext@o{#1}{#2}{#3}}%
4770     {\bbl@footnotetext@x{#1}{#2}{#3}}}]
4771 \long\def\bbl@footnotetext@x#1#2#3#4{%
4772   \bgroup
4773     \select@language@x{\bbl@main@language}%
4774     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4775   \egroup}
4776 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4777   \bgroup
4778     \select@language@x{\bbl@main@language}%
4779     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4780   \egroup}
4781 \def\BabelFootnote#1#2#3#4{%
4782   \ifx\bbl@fn@footnote\undefined
4783     \let\bbl@fn@footnote\footnote
4784   \fi
4785   \ifx\bbl@fn@footnotetext\undefined
4786     \let\bbl@fn@footnotetext\footnotetext
4787   \fi
4788   \bbl@ifblank{#2}%
4789   {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4790   \@namedef{\bbl@stripslash#1text}%
4791     {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4792   {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4793   \@namedef{\bbl@stripslash#1text}%
4794     {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}]
4795 \fi
4796 <</Footnote changes>>

```

Now, the code.

```

4797 (*xetex)
4798 \def\BabelStringsDefault{unicode}
4799 \let\xebbl@stop\relax
4800 \AddBabelHook{xetex}{encodedcommands}{%
4801   \def\bbl@tempa{#1}%
4802   \ifx\bbl@tempa@empty
4803     \XeTeXinputencoding"bytes"%
4804   \else
4805     \XeTeXinputencoding"#1"%
4806   \fi
4807   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4808 \AddBabelHook{xetex}{stopcommands}{%
4809   \xebbl@stop
4810   \let\xebbl@stop\relax}
4811 \def\bbl@intraspace#1 #2 #3\@@{%
4812   \bbl@csarg\gdef{xeisp@\languagename}%
4813     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4814 \def\bbl@intrapenalty#1\@@{%
4815   \bbl@csarg\gdef{xeipn@\languagename}%
4816     {\XeTeXlinebreakpenalty #1\relax}}

```

```

4817 \def\bb@provide@intraspace{%
4818   \bb@xin@{/s}{/\bb@cl{lbrk}}%
4819   \ifin@ \else \bb@xin@{/c}{/\bb@cl{lbrk}} \fi
4820   \ifin@
4821     \bb@ifunset{bb@intsp@\languagename}{}%
4822     {\expandafter\ifx\cname bb@intsp@\languagename\endcsname\@empty\else
4823       \ifx\bb@KVP@intraspace\@nnil
4824         \bb@exp{%
4825           \\bb@intraspace\bb@cl{intsp}\\\@}%
4826         \fi
4827         \ifx\bb@KVP@intrapenalty\@nnil
4828           \bb@intrapenalty0\@@
4829         \fi
4830       \fi
4831       \ifx\bb@KVP@intraspace\@nnil\else % We may override the ini
4832         \expandafter\bb@intraspace\bb@KVP@intraspace\@@
4833       \fi
4834       \ifx\bb@KVP@intrapenalty\@nnil\else
4835         \expandafter\bb@intrapenalty\bb@KVP@intrapenalty\@@
4836       \fi
4837     \bb@exp{%
4838       % TODO. Execute only once (but redundant):
4839       \\bb@add<extras\languagename>{%
4840         \XeTeXlinebreaklocale "\bb@cl{tbc}"%
4841         \<bb@xeisp@\languagename>%
4842         \<bb@xeipn@\languagename>%
4843         \\bb@tglobal<extras\languagename>%
4844         \\bb@add<noextras\languagename>{%
4845           \XeTeXlinebreaklocale ""}%
4846         \\bb@tglobal<noextras\languagename>%
4847       \ifx\bb@ispacesize\undefined
4848         \gdef\bb@ispacesize{\bb@cl{xeisp}}%
4849       \ifx\AtBeginDocument\@notprerr
4850         \expandafter\@secondoftwo % to execute right now
4851       \fi
4852       \AtBeginDocument{\bb@patchfont{\bb@ispacesize}}%
4853     \fi}%
4854 \fi}
4855 \ifx\DisableBabelHook\@undefined\endinput\fi
4856 \AddBabelHook{babel - fontspec}{afterextras}{\bb@switchfont}
4857 \AddBabelHook{babel - fontspec}{beforestart}{\bb@ccheckstdfonts}
4858 \DisableBabelHook{babel - fontspec}
4859 <<Font selection>>
4860 \def\bb@provide@extra#1{}

```

10 Support for interchar

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

```

4861 \ifnum\xe@alloc@intercharclass<\thr@
4862   \xe@alloc@intercharclass\thr@@
4863 \fi
4864 \chardef\bb@xe@class@default@=\z@
4865 \chardef\bb@xe@class@cjkideogram@=\@ne
4866 \chardef\bb@xe@class@cjkleftpunctuation@=\tw@
4867 \chardef\bb@xe@class@cjkrightpunctuation@=\thr@@
4868 \chardef\bb@xe@class@boundary@=4095
4869 \chardef\bb@xe@class@ignore@=4096

```

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

and after `\bbl@upto`, which is the previous char negated, as a flag to mark a range.

```

4870 \AddBabelHook{babel-interchar}{beforeextras}{%
4871   \@nameuse{bbl@xechars@\languagename}}
4872 \DisableBabelHook{babel-interchar}
4873 \protected\def\bbl@charclass#1{%
4874   \ifnum\count@<\z@
4875     \count@-\count@
4876     \loop
4877       \bbl@exp{%
4878         \\babel@savevariable{\XeTeXcharclass`Uchar\count@}}%
4879         \XeTeXcharclass\count@ \bbl@tempc
4880         \ifnum\count@<`#1\relax
4881         \advance\count@\@ne
4882       \repeat
4883   \else
4884     \babel@savevariable{\XeTeXcharclass`#1}%
4885     \XeTeXcharclass`#1 \bbl@tempc
4886   \fi
4887   \count@`#1\relax}

```

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

```

4888 \newcommand\babelcharclass[3]{%
4889   \EnableBabelHook{babel-interchar}%
4890   \bbl@csarg\newXeTeXintercharclass{xeclasse@#2@#1}%
4891   \def\bbl@tempb##1{%
4892     \ifx##1@empty\else
4893       \ifx##1-%
4894         \bbl@upto
4895       \else
4896         \bbl@charclass{%
4897           \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
4898       \fi
4899       \expandafter\bbl@tempb
4900     \fi}%
4901   \bbl@ifunset{bbl@xechars@#1}%
4902     {\toks@{%
4903       \babel@savevariable\XeTeXinterchartokenstate
4904       \XeTeXinterchartokenstate\@ne
4905     }}%
4906     {\toks@\expandafter\expandafter\expandafter{%
4907       \csname bbl@xechars@#1\endcsname}}%
4908   \bbl@csarg\edef{xechars@#1}{%
4909     \the\toks@
4910     \bbl@usingxeclasse\csname bbl@xeclasse@#2@#1\endcsname
4911     \bbl@tempb#3\@empty}}
4912 \protected\def\bbl@usingxeclasse#1{\count@\z@ \let\bbl@tempc#1}
4913 \protected\def\bbl@upto{%
4914   \ifnum\count@>\z@
4915     \advance\count@\@ne
4916     \count@-\count@
4917   \else\ifnum\count@=\z@
4918     \bbl@charclass{-}%
4919   \else
4920     \bbl@error{Double hyphens aren't allowed in \string\babelcharclass\\%
4921       because it's potentially ambiguous}%
4922     {See the manual for further info}%
4923   \fi\fi}

```

And finally, the command with the code to be inserted. If the language doesn't define a class, then

use the global one, as defined above. For the definition there is a intermediate macro, which can be 'disabled' with `\bbl@ic@<label>@<lang>`.

```

4924 \newcommand\babelinterchar[5][[]]{%
4925   \let\bbl@kv@label\@empty
4926   \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}%
4927   \@namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
4928   {\ifnum\language=\@nohyphenation
4929     \expandafter\@gobble
4930     \else
4931     \expandafter\@firstofone
4932     \fi
4933   {#5}}%
4934 \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone
4935 \bbl@exp{\bbl@for\bbl@tempa{\zap@space#3 \@empty}}{%
4936   \bbl@exp{\bbl@for\bbl@tempb{\zap@space#4 \@empty}}{%
4937     \XeTeXinterchartoks
4938     \@nameuse{bbl@xeiclass@\bbl@tempa @%
4939       \bbl@ifunset{bbl@xeiclass@\bbl@tempa @#2}{#2}}
4940     \@nameuse{bbl@xeiclass@\bbl@tempb @%
4941       \bbl@ifunset{bbl@xeiclass@\bbl@tempb @#2}{#2}}
4942     = \expandafter{%
4943       \csname bbl@ic@\bbl@kv@label @#2\expandafter\endcsname
4944       \csname\zap@space bbl@xeinter@\bbl@kv@label
4945         @#3@#4@#2 \@empty\endcsname}}}}
4946 \DeclareRobustCommand\enablelocaleinterchar[1]{%
4947   \bbl@ifunset{bbl@ic@#1@\language}%
4948   {\bbl@error
4949     { '#1' for '\language' cannot be enabled.\\%
4950     Maybe there is a typo.}%
4951     {See the manual for further details.}}%
4952   {\bbl@csarg\let{ic@#1@\language}\@firstofone}}
4953 \DeclareRobustCommand\disablelocaleinterchar[1]{%
4954   \bbl@ifunset{bbl@ic@#1@\language}%
4955   {\bbl@error
4956     { '#1' for '\language' cannot be disabled.\\%
4957     Maybe there is a typo.}%
4958     {See the manual for further details.}}%
4959   {\bbl@csarg\let{ic@#1@\language}\@gobble}}
4960 \end{document}

```

10.1 Layout

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

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

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

```

4961 (*xetex | texet)
4962 \providecommand\bbl@provide@intraspace{}
4963 \bbl@trace{Redefinitions for bidi layout}
4964 \def\bbl@sspre@caption{%
4965   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4966 \ifx\bbl@opt@layout\@nnil\else % if layout=..
4967 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4968 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4969 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4970 \def\@hangfrom#1{%
4971   \setbox\@tempboxa\hbox{#1}}%
4972 \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4973 \noindent\box\@tempboxa}
4974 \def\raggedright{%

```

```

4975 \let\\@centercr
4976 \bbl@startskip\z@skip
4977 \@rightskip\@flushglue
4978 \bbl@endskip\@rightskip
4979 \parindent\z@
4980 \parfillskip\bbl@startskip}
4981 \def\raggedleft{%
4982 \let\\@centercr
4983 \bbl@startskip\@flushglue
4984 \bbl@endskip\z@skip
4985 \parindent\z@
4986 \parfillskip\bbl@endskip}
4987 \fi
4988 \IfBabelLayout{lists}
4989 {\bbl@sreplace\list
4990 {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4991 \def\bbl@listleftmargin{%
4992 \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4993 \ifcase\bbl@engine
4994 \def\labelenumii{}\theenumii}% pdftex doesn't reverse ()
4995 \def\p@enumiii{\p@enumii}\theenumii}%
4996 \fi
4997 \bbl@sreplace\@verbatim
4998 {\leftskip\@totalleftmargin}%
4999 {\bbl@startskip\textwidth
5000 \advance\bbl@startskip-\linewidth}%
5001 \bbl@sreplace\@verbatim
5002 {\rightskip\z@skip}%
5003 {\bbl@endskip\z@skip}}%
5004 {}
5005 \IfBabelLayout{contents}
5006 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
5007 \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
5008 {}
5009 \IfBabelLayout{columns}
5010 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
5011 \def\bbl@outputbox#1{%
5012 \hb@xt@\textwidth{%
5013 \hskip\columnwidth
5014 \hfil
5015 {\normalcolor\vrule \@width\columnseprule}%
5016 \hfil
5017 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
5018 \hskip-\textwidth
5019 \hb@xt@\columnwidth{\box\@outputbox \hss}%
5020 \hskip\columnsep
5021 \hskip\columnwidth}}}%
5022 {}
5023 \langle\langle Footnote changes \rangle\rangle
5024 \IfBabelLayout{footnotes}%
5025 {\BabelFootnote\footnote\languagename}{}%
5026 \BabelFootnote\localfootnote\languagename}{}%
5027 \BabelFootnote\mainfootnote}{}}
5028 {}
5029 \IfBabelLayout{counters*}%
5030 {\bbl@add\bbl@opt@layout{.counters.}%
5031 \AddToHook{shipout/before}{%
5032 \let\bbl@tempa\babelsublr
5033 \let\babelsublr\@firstofone
5034 \let\bbl@save@thepage\thepage

```

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.

```

5035     \protected@edef\thepage{\thepage}%
5036     \let\babelsublr\bbl@tempa}%
5037     \AddToHook{shipout/after}{%
5038         \let\thepage\bbl@save@thepage}}{}
5039 \IfBabelLayout{counters}%
5040     {\let\bbl@latinarabic=@arabic
5041     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5042     \let\bbl@asciroman=@roman
5043     \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5044     \let\bbl@asciiRoman=@Roman
5045     \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
5046 \fi % end if layout
5047 </xetex | texxtet)

```

10.2 8-bit TeX

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

```

5048 (*texxtet)
5049 \def\bbl@provide@extra#1{%
5050     % == auto-select encoding ==
5051     \ifx\bbl@encoding@select@off\@empty\else
5052         \bbl@ifunset{\bbl@encoding@#1}%
5053             {\def\@elt##1{,##1},}%
5054             \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5055             \count@\z@
5056             \bbl@foreach\bbl@tempe{%
5057                 \def\bbl@tempd{##1}% Save last declared
5058                 \advance\count@\@ne}%
5059             \ifnum\count@>\@ne
5060                 \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5061                 \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
5062                 \bbl@replace\bbl@tempa{ },{,}%
5063                 \global\bbl@csarg\let{encoding@#1}\@empty
5064                 \bbl@xin@{,\bbl@tempd,}{,\bbl@tempa,}%
5065                 \ifin@else % if main encoding included in ini, do nothing
5066                     \let\bbl@tempb\relax
5067                     \bbl@foreach\bbl@tempa{%
5068                         \ifx\bbl@tempb\relax
5069                             \bbl@xin@{,##1,}{,\bbl@tempe,}%
5070                             \ifin@def\bbl@tempb{##1}\fi
5071                         \fi}%
5072                 \ifx\bbl@tempb\relax\else
5073                     \bbl@exp{%
5074                         \global\<\bbl@add>\<\bbl@preextras@#1>{\<\bbl@encoding@#1>}%
5075                         \gdef\<\bbl@encoding@#1>{%
5076                             \\babel@save\\f@encoding
5077                             \\bbl@add\\originalTeX{\\selectfont}%
5078                             \\fontencoding{\bbl@tempb}%
5079                             \\selectfont}}%
5080                     \fi
5081                 \fi
5082             \fi}%
5083         }%
5084     \fi}
5085 </texxtet)

```

10.3 LuaTeX

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

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

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

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

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

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

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

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

```

5086 (*luatex)
5087 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
5088 \bbl@trace{Read language.dat}
5089 \ifx\bbl@readstream\undefined
5090 \csname newread\endcsname\bbl@readstream
5091 \fi
5092 \begingroup
5093 \toks@{}
5094 \count@z@ % 0=start, 1=0th, 2=normal
5095 \def\bbl@process@line#1#2 #3 #4 {%
5096   \ifx=#1%
5097     \bbl@process@synonym{#2}%
5098   \else
5099     \bbl@process@language{#1#2}{#3}{#4}%
5100   \fi
5101   \ignorespaces}
5102 \def\bbl@manylang{%
5103   \ifnum\bbl@last>\@ne
5104     \bbl@info{Non-standard hyphenation setup}%
5105   \fi
5106   \let\bbl@manylang\relax}
5107 \def\bbl@process@language#1#2#3{%
5108   \ifcase\count@
5109     \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
5110   \or
5111     \count@\tw@
5112   \fi
5113   \ifnum\count@=\tw@
5114     \expandafter\addlanguage\csname l@#1\endcsname
5115     \language\allocationnumber
5116     \chardef\bbl@last\allocationnumber
5117     \bbl@manylang
5118     \let\bbl@elt\relax
5119   \xdef\bbl@languages{%
5120     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%

```

```

5121 \fi
5122 \the\toks@
5123 \toks@{}}
5124 \def\bbl@process@synonym@aux#1#2{%
5125 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
5126 \let\bbl@elt\relax
5127 \xdef\bbl@languages{%
5128 \bbl@languages\bbl@elt{#1}{#2}{}}}%
5129 \def\bbl@process@synonym#1{%
5130 \ifcase\count@
5131 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
5132 \or
5133 \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}}%
5134 \else
5135 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
5136 \fi}
5137 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
5138 \chardef\l@english\z@
5139 \chardef\l@USenglish\z@
5140 \chardef\bbl@last\z@
5141 \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
5142 \gdef\bbl@languages{%
5143 \bbl@elt{english}{0}{hyphen.tex}}%
5144 \bbl@elt{USenglish}{0}{}}
5145 \else
5146 \global\let\bbl@languages@format\bbl@languages
5147 \def\bbl@elt#1#2#3#4{% Remove all except language 0
5148 \ifnum#2>\z@\else
5149 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
5150 \fi}%
5151 \xdef\bbl@languages{\bbl@languages}%
5152 \fi
5153 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}} % Define flags
5154 \bbl@languages
5155 \openin\bbl@readstream=language.dat
5156 \ifeof\bbl@readstream
5157 \bbl@warning{I couldn't find language.dat. No additional\\%
5158 patterns loaded. Reported}%
5159 \else
5160 \loop
5161 \endlinechar\m@ne
5162 \read\bbl@readstream to \bbl@line
5163 \endlinechar\^^M
5164 \if T\ifeof\bbl@readstream F\fi T\relax
5165 \ifx\bbl@line\@empty\else
5166 \edef\bbl@line{\bbl@line\space\space\space}%
5167 \expandafter\bbl@process@line\bbl@line\relax
5168 \fi
5169 \repeat
5170 \fi
5171 \closein\bbl@readstream
5172 \endgroup
5173 \bbl@trace{Macros for reading patterns files}
5174 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
5175 \ifx\babelcatcodetablenum\@undefined
5176 \ifx\newcatcodetable\@undefined
5177 \def\babelcatcodetablenum{5211}
5178 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5179 \else
5180 \newcatcodetable\babelcatcodetablenum
5181 \newcatcodetable\bbl@pattcodes
5182 \fi
5183 \else

```

```

5184 \def\bbbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5185 \fi
5186 \def\bbbl@luapatterns#1#2{%
5187 \bbbl@get@enc#1:\@@@
5188 \setbox\z@\hbox\bgroup
5189 \begingroup
5190 \savecatcodetable\babelcatcodetablenum\relax
5191 \initcatcodetable\bbbl@pattcodes\relax
5192 \catcodetable\bbbl@pattcodes\relax
5193 \catcode\#=6 \catcode\$_=3 \catcode\&=4 \catcode\^=7
5194 \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\~=13
5195 \catcode\@=11 \catcode\^^I=10 \catcode\^^J=12
5196 \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
5197 \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
5198 \catcode\`=12 \catcode\`=12 \catcode\`=12
5199 \input #1\relax
5200 \catcodetable\babelcatcodetablenum\relax
5201 \endgroup
5202 \def\bbbl@tempa{#2}%
5203 \ifx\bbbl@tempa\@empty\else
5204 \input #2\relax
5205 \fi
5206 \egroup}%
5207 \def\bbbl@patterns@lua#1{%
5208 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5209 \csname l@#1\endcsname
5210 \edef\bbbl@tempa{#1}%
5211 \else
5212 \csname l@#1:\f@encoding\endcsname
5213 \edef\bbbl@tempa{#1:\f@encoding}%
5214 \fi\relax
5215 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5216 \@ifundefined{bbbl@hyphendata@the\language}%
5217 {\def\bbbl@elt##1##2##3##4{%
5218 \ifnum##2=\csname l@bbbl@tempa\endcsname % #2=spanish, dutch:OT1...
5219 \def\bbbl@tempb{##3}%
5220 \ifx\bbbl@tempb\@empty\else % if not a synonymous
5221 \def\bbbl@tempc{{##3}{##4}}%
5222 \fi
5223 \bbbl@csarg\xdef{hyphendata@##2}{\bbbl@tempc}%
5224 \fi}%
5225 \bbbl@languages
5226 \@ifundefined{bbbl@hyphendata@the\language}%
5227 {\bbbl@info{No hyphenation patterns were set for\%
5228 language '\bbbl@tempa'. Reported}}%
5229 {\expandafter\expandafter\expandafter\bbbl@luapatterns
5230 \csname bbl@hyphendata@the\language\endcsname}}}}
5231 \endinput\fi
5232 % Here ends \ifx\AddBabelHook\@undefined
5233 % A few lines are only read by hyphen.cfg
5234 \ifx\DisableBabelHook\@undefined
5235 \AddBabelHook{luatex}{everylanguage}{%
5236 \def\process@language##1##2##3{%
5237 \def\process@line####1####2 ####3 ####4 {}}}
5238 \AddBabelHook{luatex}{loadpatterns}{%
5239 \input #1\relax
5240 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5241 {#1}{}}
5242 \AddBabelHook{luatex}{loadexceptions}{%
5243 \input #1\relax
5244 \def\bbbl@tempb##1##2{{##1}{##2}}%
5245 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5246 {\expandafter\expandafter\expandafter\bbbl@tempb

```

```

5247     \csname bbl@hyphendata@the\language\endcsname}}
5248 \endinput\fi
5249 % Here stops reading code for hyphen.cfg
5250 % The following is read the 2nd time it's loaded
5251 \begingroup % TODO - to a lua file
5252 \catcode`\%=12
5253 \catcode`\'=12
5254 \catcode`\ "=12
5255 \catcode`\:=12
5256 \directlua{
5257   Babel = Babel or {}
5258   function Babel.bytes(line)
5259     return line:gsub(".",
5260       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5261   end
5262   function Babel.begin_process_input()
5263     if luatexbase and luatexbase.add_to_callback then
5264       luatexbase.add_to_callback('process_input_buffer',
5265         Babel.bytes, 'Babel.bytes')
5266     else
5267       Babel.callback = callback.find('process_input_buffer')
5268       callback.register('process_input_buffer', Babel.bytes)
5269     end
5270   end
5271   function Babel.end_process_input ()
5272     if luatexbase and luatexbase.remove_from_callback then
5273       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5274     else
5275       callback.register('process_input_buffer', Babel.callback)
5276     end
5277   end
5278   function Babel.addpatterns(pp, lg)
5279     local lg = lang.new(lg)
5280     local pats = lang.patterns(lg) or ''
5281     lang.clear_patterns(lg)
5282     for p in pp:gmatch('[^%s]+') do
5283       ss = ''
5284       for i in string.utfcharacters(p:gsub('%d', '')) do
5285         ss = ss .. '%d?' .. i
5286       end
5287       ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
5288       ss = ss:gsub('%.%d%?$', '%%.')
5289       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5290       if n == 0 then
5291         tex.sprint(
5292           [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5293           .. p .. [[]])
5294         pats = pats .. ' ' .. p
5295       else
5296         tex.sprint(
5297           [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5298           .. p .. [[]])
5299       end
5300     end
5301     lang.patterns(lg, pats)
5302   end
5303   Babel.characters = Babel.characters or {}
5304   Babel.ranges = Babel.ranges or {}
5305   function Babel.hlist_has_bidi(head)
5306     local has_bidi = false
5307     local ranges = Babel.ranges
5308     for item in node.traverse(head) do
5309       if item.id == node.id'glyph' then

```



```

5310     local itemchar = item.char
5311     local chardata = Babel.characters[itemchar]
5312     local dir = chardata and chardata.d or nil
5313     if not dir then
5314         for nn, et in ipairs(ranges) do
5315             if itemchar < et[1] then
5316                 break
5317             elseif itemchar <= et[2] then
5318                 dir = et[3]
5319                 break
5320             end
5321         end
5322     end
5323     if dir and (dir == 'al' or dir == 'r') then
5324         has_bidi = true
5325     end
5326 end
5327 end
5328 return has_bidi
5329 end
5330 function Babel.set_chranges_b (script, chrng)
5331     if chrng == '' then return end
5332     texio.write('Replacing ' .. script .. ' script ranges')
5333     Babel.script_blocks[script] = {}
5334     for s, e in string.gmatch(chrng..' ', '(.)%.%.(.-)%s') do
5335         table.insert(
5336             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5337     end
5338 end
5339 function Babel.discard_sublr(str)
5340     if str:find( [[\string\indexentry]] ) and
5341        str:find( [[\string\babelsublr]] ) then
5342         str = str:gsub( [[\string\babelsublr%*(%b{})]],
5343             function(m) return m:sub(2,-2) end )
5344     end
5345     return str
5346 end
5347 }
5348 \endgroup
5349 \ifx\newattribute\undefined\else % Test for plain
5350 \newattribute\bbl@attr@locale
5351 \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5352 \AddBabelHook{luatex}{beforeextras}{%
5353     \setattribute\bbl@attr@locale\localeid}
5354 \fi
5355 \def\BabelStringsDefault{unicode}
5356 \let\luabbl@stop\relax
5357 \AddBabelHook{luatex}{encodedcommands}{%
5358     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5359     \ifx\bbl@tempa\bbl@tempb\else
5360         \directlua{Babel.begin_process_input()}%
5361         \def\luabbl@stop{%
5362             \directlua{Babel.end_process_input()}%
5363         \fi}%
5364 \AddBabelHook{luatex}{stopcommands}{%
5365     \luabbl@stop
5366     \let\luabbl@stop\relax}
5367 \AddBabelHook{luatex}{patterns}{%
5368     \@ifundefined{bbl@hyphendata@the\language}%
5369     {\def\bbl@elt##1##2##3##4{%
5370         \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5371         \def\bbl@tempb{##3}%
5372         \ifx\bbl@tempb\empty\else % if not a synonymous

```

```

5373         \def\bb@tempc{##3}##4}%
5374         \fi
5375         \bb@csarg\xdef{hyphendata@##2}{\bb@tempc}%
5376         \fi}%
5377     \bb@languages
5378     \@ifundefined{bb@hyphendata@the\language}%
5379     {\bb@info{No hyphenation patterns were set for\%
5380         language '#2'. Reported}}%
5381     {\expandafter\expandafter\expandafter\bb@luapatterns
5382         \csname bb@hyphendata@the\language\endcsname}}}%
5383     \@ifundefined{bb@patterns@}{}%
5384     \begingroup
5385         \bb@xin@{,\number\language,}{,\bb@pttnlist}%
5386         \ifin\else
5387             \ifx\bb@patterns@\empty\else
5388                 \directlua{ Babel.addpatterns(
5389                     [[\bb@patterns@]], \number\language) }%
5390             \fi
5391             \@ifundefined{bb@patterns@#1}%
5392             \@empty
5393             {\directlua{ Babel.addpatterns(
5394                 [[\space\csname bb@patterns@#1\endcsname]],
5395                 \number\language) }}%
5396             \xdef\bb@pttnlist{\bb@pttnlist\number\language,}%
5397         \fi
5398     \endgroup}%
5399     \bb@exp{%
5400         \bb@ifunset{bb@prehc@\language}{}%
5401         {\bb@ifblank{\bb@cs{prehc@\language}}}%
5402         {\prehyphenchar=\bb@cl{prehc}\relax}}

```

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

```

5403 \@onlypreamble\babelpatterns
5404 \AtEndOfPackage{%
5405     \newcommand\babelpatterns[2][\@empty]{%
5406         \ifx\bb@patterns@\relax
5407             \let\bb@patterns@\empty
5408             \fi
5409         \ifx\bb@pttnlist@\empty\else
5410             \bb@warning{%
5411                 You must not intermingle \string\selectlanguage\space and\%
5412                 \string\babelpatterns\space or some patterns will not\%
5413                 be taken into account. Reported}%
5414             \fi
5415         \ifx\@empty#1%
5416             \protected@edef\bb@patterns@{\bb@patterns@\space#2}%
5417         \else
5418             \edef\bb@tempb{\zap@space#1 \@empty}%
5419             \bb@for\bb@tempa\bb@tempb{%
5420                 \bb@fixname\bb@tempa
5421                 \bb@iflanguage\bb@tempa{%
5422                     \bb@csarg\protected@edef{patterns@\bb@tempa}{%
5423                         \@ifundefined{bb@patterns@\bb@tempa}%
5424                         \@empty
5425                         {\csname bb@patterns@\bb@tempa\endcsname\space}%
5426                         #2}}}%
5427             \fi}}

```

10.4 Southeast Asian scripts

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

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

```

5428% TODO - to a lua file
5429 \directlua{
5430   Babel = Babel or {}
5431   Babel.linebreaking = Babel.linebreaking or {}
5432   Babel.linebreaking.before = {}
5433   Babel.linebreaking.after = {}
5434   Babel.locale = {} % Free to use, indexed by \localeid
5435   function Babel.linebreaking.add_before(func, pos)
5436     tex.print([[ \noexpand\csname bbl@luaohyphenate\endcsname]])
5437     if pos == nil then
5438       table.insert(Babel.linebreaking.before, func)
5439     else
5440       table.insert(Babel.linebreaking.before, pos, func)
5441     end
5442   end
5443   function Babel.linebreaking.add_after(func)
5444     tex.print([[ \noexpand\csname bbl@luaohyphenate\endcsname]])
5445     table.insert(Babel.linebreaking.after, func)
5446   end
5447 }
5448 \def\bbl@intraspace#1 #2 #3\@@{
5449   \directlua{
5450     Babel = Babel or {}
5451     Babel.intraspaces = Babel.intraspaces or {}
5452     Babel.intraspaces['\csname bbl@sbcpr@languagenam\endcsname'] = %
5453       {b = #1, p = #2, m = #3}
5454     Babel.locale_props[\the\localeid].intraspace = %
5455       {b = #1, p = #2, m = #3}
5456   }}
5457 \def\bbl@intrapenalty#1\@@{
5458   \directlua{
5459     Babel = Babel or {}
5460     Babel.intrapenalties = Babel.intrapenalties or {}
5461     Babel.intrapenalties['\csname bbl@sbcpr@languagenam\endcsname'] = #1
5462     Babel.locale_props[\the\localeid].intrapenalty = #1
5463   }}
5464 \begingroup
5465 \catcode`\%=12
5466 \catcode`\^=14
5467 \catcode`\'=12
5468 \catcode`\-=12
5469 \gdef\bbl@seaintraspace{
5470   \let\bbl@seaintraspace\relax
5471   \directlua{
5472     Babel = Babel or {}
5473     Babel.sea_enabled = true
5474     Babel.sea_ranges = Babel.sea_ranges or {}
5475     function Babel.set_chranges (script, chrng)
5476       local c = 0
5477       for s, e in string.gmatch(chrng..' ', '(.)%.%.(-)%s') do
5478         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5479         c = c + 1
5480       end
5481     end
5482     function Babel.sea_disc_to_space (head)
5483       local sea_ranges = Babel.sea_ranges
5484       local last_char = nil
5485       local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5486       for item in node.traverse(head) do
5487         local i = item.id

```

```

5488     if i == node.id'glyph' then
5489         last_char = item
5490     elseif i == 7 and item.subtype == 3 and last_char
5491         and last_char.char > 0x0C99 then
5492         quad = font.getfont(last_char.font).size
5493         for lg, rg in pairs(sea_ranges) do
5494             if last_char.char > rg[1] and last_char.char < rg[2] then
5495                 lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyril
5496                 local intraspace = Babel.intraspaces[lg]
5497                 local intrapenalty = Babel.intrapenalties[lg]
5498                 local n
5499                 if intrapenalty ~= 0 then
5500                     n = node.new(14, 0)  ^% penalty
5501                     n.penalty = intrapenalty
5502                     node.insert_before(head, item, n)
5503                 end
5504                 n = node.new(12, 13)  ^% (glue, spaceskip)
5505                 node.setglue(n, intraspace.b * quad,
5506                             intraspace.p * quad,
5507                             intraspace.m * quad)
5508                 node.insert_before(head, item, n)
5509                 node.remove(head, item)
5510             end
5511         end
5512     end
5513 end
5514 end
5515 }^^
5516 \bbl@luahyphenate}

```

10.5 CJK line breaking

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

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

```

5517 \catcode`\%=14
5518 \gdef\bbl@cjkintraspaces{%
5519   \let\bbl@cjkintraspaces\relax
5520   \directlua{
5521     Babel = Babel or {}
5522     require('babel-data-cjk.lua')
5523     Babel.cjk_enabled = true
5524     function Babel.cjk_linebreak(head)
5525       local GLYPH = node.id'glyph'
5526       local last_char = nil
5527       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5528       local last_class = nil
5529       local last_lang = nil
5530
5531       for item in node.traverse(head) do
5532         if item.id == GLYPH then
5533
5534           local lang = item.lang
5535
5536           local LOCALE = node.get_attribute(item,
5537             Babel.attr_locale)
5538           local props = Babel.locale_props[LOCALE]
5539
5540           local class = Babel.cjk_class[item.char].c
5541

```

```

5542     if props.cjk_quotes and props.cjk_quotes[item.char] then
5543         class = props.cjk_quotes[item.char]
5544     end
5545
5546     if class == 'cp' then class = 'cl' end % )) as CL
5547     if class == 'id' then class = 'I' end
5548
5549     local br = 0
5550     if class and last_class and Babel.cjk_breaks[last_class][class] then
5551         br = Babel.cjk_breaks[last_class][class]
5552     end
5553
5554     if br == 1 and props.linebreak == 'c' and
5555         lang ~= \the\l@nohyphenation\space and
5556         last_lang ~= \the\l@nohyphenation then
5557         local intrapenalty = props.intrapenalty
5558         if intrapenalty ~= 0 then
5559             local n = node.new(14, 0)      % penalty
5560             n.penalty = intrapenalty
5561             node.insert_before(head, item, n)
5562         end
5563         local intraspace = props.intraspace
5564         local n = node.new(12, 13)      % (glue, spaceskip)
5565         node.setglue(n, intraspace.b * quad,
5566                     intraspace.p * quad,
5567                     intraspace.m * quad)
5568         node.insert_before(head, item, n)
5569     end
5570
5571     if font.getfont(item.font) then
5572         quad = font.getfont(item.font).size
5573     end
5574     last_class = class
5575     last_lang = lang
5576     else % if penalty, glue or anything else
5577         last_class = nil
5578     end
5579     end
5580     lang.hyphenate(head)
5581 end
5582 }%
5583 \bbl@luaohyphenate}
5584 \gdef\bbl@luaohyphenate{%
5585 \let\bbl@luaohyphenate\relax
5586 \directlua{
5587     luatexbase.add_to_callback('hyphenate',
5588     function (head, tail)
5589         if Babel.linebreaking.before then
5590             for k, func in ipairs(Babel.linebreaking.before) do
5591                 func(head)
5592             end
5593         end
5594         if Babel.cjk_enabled then
5595             Babel.cjk_linebreak(head)
5596         end
5597         lang.hyphenate(head)
5598         if Babel.linebreaking.after then
5599             for k, func in ipairs(Babel.linebreaking.after) do
5600                 func(head)
5601             end
5602         end
5603         if Babel.sea_enabled then
5604             Babel.sea_disc_to_space(head)

```

```

5605     end
5606     end,
5607     'Babel.hyphenate')
5608 }
5609 }
5610 \endgroup
5611 \def\bbbl@provide@intraspace{%
5612 \bbbl@ifunset{\bbbl@intsp@\languagename}{}%
5613 {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\empty\else
5614 \bbbl@xin@{/c}{/\bbbl@cl{\lnbrk}}}%
5615 \ifin@ % cjk
5616 \bbbl@cjk@intraspace
5617 \directlua{
5618     Babel = Babel or {}
5619     Babel.locale_props = Babel.locale_props or {}
5620     Babel.locale_props[\the\localeid].linebreak = 'c'
5621 }%
5622 \bbbl@exp{\bbbl@intraspace\bbbl@cl{intsp}}\@@}%
5623 \ifx\bbbl@KVP@intrapenalty\@nnil
5624 \bbbl@intrapenalty0\@@
5625 \fi
5626 \else % sea
5627 \bbbl@sea@intraspace
5628 \bbbl@exp{\bbbl@intraspace\bbbl@cl{intsp}}\@@}%
5629 \directlua{
5630     Babel = Babel or {}
5631     Babel.sea_ranges = Babel.sea_ranges or {}
5632     Babel.set_chranges('\bbbl@cl{sbcpr}',
5633                       '\bbbl@cl{chrng}')
5634 }%
5635 \ifx\bbbl@KVP@intrapenalty\@nnil
5636 \bbbl@intrapenalty0\@@
5637 \fi
5638 \fi
5639 \fi
5640 \ifx\bbbl@KVP@intrapenalty\@nnil\else
5641 \expandafter\bbbl@intrapenalty\bbbl@KVP@intrapenalty\@@
5642 \fi}}

```

10.6 Arabic justification

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

```

5643 \ifnum\bbbl@bidimode>100 \ifnum\bbbl@bidimode<200
5644 \def\bblar@chars{%
5645     0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5646     0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5647     0640,0641,0642,0643,0644,0645,0646,0647,0649}
5648 \def\bblar@elongated{%
5649     0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5650     063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5651     0649,064A}
5652 \begingroup
5653 \catcode`_=11 \catcode`:=11
5654 \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5655 \endgroup
5656 \gdef\bbbl@arabicjust{% TODO. Allow for several locales.
5657 \let\bbbl@arabicjust\relax
5658 \newattribute\bblar@kashida
5659 \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5660 \bblar@kashida=\z@
5661 \bbbl@patchfont{\bbbl@parsejalt}}%
5662 \directlua{

```

```

5663 Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5664 Babel.arabic.elong_map[\the\localeid] = {}
5665 luatexbase.add_to_callback('post_linebreak_filter',
5666   Babel.arabic.justify, 'Babel.arabic.justify')
5667 luatexbase.add_to_callback('hpack_filter',
5668   Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5669 }%

```

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

```

5670 \def\bblar@fetchjalt#1#2#3#4{%
5671   \bbl@exp{\bbl@foreach{#1}}{%
5672     \bbl@ifunset{bblar@JE@##1}%
5673     {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"##1#2}}%
5674     {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"\@nameuse{bblar@JE@##1}#2}}%
5675     \directlua{%
5676       local last = nil
5677       for item in node.traverse(tex.box[0].head) do
5678         if item.id == node.id'glyph' and item.char > 0x600 and
5679           not (item.char == 0x200D) then
5680           last = item
5681         end
5682       end
5683       Babel.arabic.#3['##1#4'] = last.char
5684     }}

```

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?

```

5685 \gdef\bbl@parsejalt{%
5686   \ifx\addfontfeature\@undefined\else
5687     \bbl@xin@{/e}{/\bbl@cl{\lnbrk}}%
5688     \ifin@
5689     \directlua{%
5690       if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5691         Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5692         tex.print([[string\curname\space bbl@parsejalti\endcurname]])
5693       end
5694     }%
5695     \fi
5696   \fi}
5697 \gdef\bbl@parsejalti{%
5698   \begingroup
5699     \let\bbl@parsejalt\relax % To avoid infinite loop
5700     \edef\bbl@tempb{\fontid\font}%
5701     \bblar@nofswarn
5702     \bblar@fetchjalt\bblar@elongated{}{from}}%
5703     \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
5704     \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
5705     \addfontfeature{RawFeature+=jalt}%
5706     % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5707     \bblar@fetchjalt\bblar@elongated{}{dest}}%
5708     \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
5709     \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
5710     \directlua{%
5711       for k, v in pairs(Babel.arabic.from) do
5712         if Babel.arabic.dest[k] and
5713           not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5714           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5715             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5716         end
5717       end
5718     }%
5719   \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```

5720 \beginingroup
5721 \catcode`#=11
5722 \catcode`~ =11
5723 \directlua{
5724
5725 Babel.arabic = Babel.arabic or {}
5726 Babel.arabic.from = {}
5727 Babel.arabic.dest = {}
5728 Babel.arabic.justify_factor = 0.95
5729 Babel.arabic.justify_enabled = true
5730 Babel.arabic.kashida_limit = -1
5731
5732 function Babel.arabic.justify(head)
5733   if not Babel.arabic.justify_enabled then return head end
5734   for line in node.traverse_id(node.id'hlist', head) do
5735     Babel.arabic.justify_hlist(head, line)
5736   end
5737   return head
5738 end
5739
5740 function Babel.arabic.justify_hbox(head, gc, size, pack)
5741   local has_inf = false
5742   if Babel.arabic.justify_enabled and pack == 'exactly' then
5743     for n in node.traverse_id(12, head) do
5744       if n.stretch_order > 0 then has_inf = true end
5745     end
5746     if not has_inf then
5747       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5748     end
5749   end
5750   return head
5751 end
5752
5753 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5754   local d, new
5755   local k_list, k_item, pos_inline
5756   local width, width_new, full, k_curr, wt_pos, goal, shift
5757   local subst_done = false
5758   local elong_map = Babel.arabic.elong_map
5759   local cnt
5760   local last_line
5761   local GLYPH = node.id'glyph'
5762   local KASHIDA = Babel.attr_kashida
5763   local LOCALE = Babel.attr_locale
5764
5765   if line == nil then
5766     line = {}
5767     line.glue_sign = 1
5768     line.glue_order = 0
5769     line.head = head
5770     line.shift = 0
5771     line.width = size
5772   end
5773
5774   % Exclude last line. todo. But-- it discards one-word lines, too!
5775   % ? Look for glue = 12:15
5776   if (line.glue_sign == 1 and line.glue_order == 0) then
5777     elongs = {} % Stores elongated candidates of each line
5778     k_list = {} % And all letters with kashida
5779     pos_inline = 0 % Not yet used
5780
5781     for n in node.traverse_id(GLYPH, line.head) do
5782       pos_inline = pos_inline + 1 % To find where it is. Not used.

```



```

5783
5784 % Elongated glyphs
5785 if elong_map then
5786     local locale = node.get_attribute(n, LOCALE)
5787     if elong_map[locale] and elong_map[locale][n.font] and
5788         elong_map[locale][n.font][n.char] then
5789         table.insert(elongs, {node = n, locale = locale} )
5790         node.set_attribute(n.prev, KASHIDA, 0)
5791     end
5792 end
5793
5794 % Tatwil
5795 if Babel.kashida_wts then
5796     local k_wt = node.get_attribute(n, KASHIDA)
5797     if k_wt > 0 then % todo. parameter for multi inserts
5798         table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5799     end
5800 end
5801
5802 end % of node.traverse_id
5803
5804 if #elongs == 0 and #k_list == 0 then goto next_line end
5805 full = line.width
5806 shift = line.shift
5807 goal = full * Babel.arabic.justify_factor % A bit crude
5808 width = node.dimensions(line.head) % The 'natural' width
5809
5810 % == Elongated ==
5811 % Original idea taken from 'chickenize'
5812 while (#elongs > 0 and width < goal) do
5813     subst_done = true
5814     local x = #elongs
5815     local curr = elongs[x].node
5816     local oldchar = curr.char
5817     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5818     width = node.dimensions(line.head) % Check if the line is too wide
5819     % Substitute back if the line would be too wide and break:
5820     if width > goal then
5821         curr.char = oldchar
5822         break
5823     end
5824     % If continue, pop the just substituted node from the list:
5825     table.remove(elongs, x)
5826 end
5827
5828 % == Tatwil ==
5829 if #k_list == 0 then goto next_line end
5830
5831 width = node.dimensions(line.head) % The 'natural' width
5832 k_curr = #k_list % Traverse backwards, from the end
5833 wt_pos = 1
5834
5835 while width < goal do
5836     subst_done = true
5837     k_item = k_list[k_curr].node
5838     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5839         d = node.copy(k_item)
5840         d.char = 0x0640
5841         d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
5842         d.xoffset = 0
5843         line.head, new = node.insert_after(line.head, k_item, d)
5844         width_new = node.dimensions(line.head)
5845         if width > goal or width == width_new then

```

```

5846         node.remove(line.head, new) % Better compute before
5847         break
5848     end
5849     if Babel.fix_diacr then
5850         Babel.fix_diacr(k_item.next)
5851     end
5852     width = width_new
5853 end
5854 if k_curr == 1 then
5855     k_curr = #k_list
5856     wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5857 else
5858     k_curr = k_curr - 1
5859 end
5860 end
5861
5862 % Limit the number of tatweel by removing them. Not very efficient,
5863 % but it does the job in a quite predictable way.
5864 if Babel.arabic.kashida_limit > -1 then
5865     cnt = 0
5866     for n in node.traverse_id(GLYPH, line.head) do
5867         if n.char == 0x0640 then
5868             cnt = cnt + 1
5869             if cnt > Babel.arabic.kashida_limit then
5870                 node.remove(line.head, n)
5871             end
5872         else
5873             cnt = 0
5874         end
5875     end
5876 end
5877
5878 ::next_line::
5879
5880 % Must take into account marks and ins, see luatex manual.
5881 % Have to be executed only if there are changes. Investigate
5882 % what's going on exactly.
5883 if subst_done and not gc then
5884     d = node.hpack(line.head, full, 'exactly')
5885     d.shift = shift
5886     node.insert_before(head, line, d)
5887     node.remove(head, line)
5888 end
5889 end % if process line
5890 end
5891 }
5892 \endgroup
5893 \fi\fi % ends Arabic just block: \ifnum\bbl@bidimode>100...

```

10.7 Common stuff

```

5894 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5895 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ccheckstdfonts}
5896 \DisableBabelHook{babel-fontspec}
5897 <<{Font selection}>>

```

10.8 Automatic fonts and ids switching

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

latter table a key starting with / maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```
5898% TODO - to a lua file
5899 \directlua{
5900 Babel.script_blocks = {
5901   ['dflt'] = {},
5902   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5903             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5904   ['Armn'] = {{0x0530, 0x058F}},
5905   ['Beng'] = {{0x0980, 0x09FF}},
5906   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5907   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5908   ['Cyrł'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5909             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5910   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5911   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5912             {0xAB00, 0xAB2F}},
5913   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5914   % Don't follow strictly Unicode, which places some Coptic letters in
5915   % the 'Greek and Coptic' block
5916   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5917   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5918             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5919             {0xF900, 0FAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5920             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5921             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5922             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5923   ['Hebr'] = {{0x0590, 0x05FF}},
5924   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5925             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5926   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5927   ['Knda'] = {{0x0C80, 0x0CFF}},
5928   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5929             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5930             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5931   ['Lao'] = {{0x0E80, 0x0EFF}},
5932   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5933             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5934             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5935   ['Mahj'] = {{0x1150, 0x117F}},
5936   ['Mlym'] = {{0x0D00, 0x0D7F}},
5937   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5938   ['Orya'] = {{0x0B00, 0x0B7F}},
5939   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x11E0, 0x11FF}},
5940   ['Sycr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5941   ['Taml'] = {{0x0B80, 0x0BFF}},
5942   ['Telu'] = {{0x0C00, 0x0C7F}},
5943   ['Tfng'] = {{0x2D30, 0x2D7F}},
5944   ['Thai'] = {{0x0E00, 0x0E7F}},
5945   ['Tibt'] = {{0x0F00, 0x0FFF}},
5946   ['Vaii'] = {{0xA500, 0xA63F}},
5947   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5948 }
5949
5950 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrł
5951 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5952 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5953
5954 function Babel.locale_map(head)
5955   if not Babel.locale_mapped then return head end
5956
5957   local LOCALE = Babel.attr_locale
5958   local GLYPH = node.id('glyph')
```

```

5959 local inmath = false
5960 local toloc_save
5961 for item in node.traverse(head) do
5962   local toloc
5963   if not inmath and item.id == GLYPH then
5964     % Optimization: build a table with the chars found
5965     if Babel.chr_to_loc[item.char] then
5966       toloc = Babel.chr_to_loc[item.char]
5967     else
5968       for lc, maps in pairs(Babel.loc_to_scr) do
5969         for _, rg in pairs(maps) do
5970           if item.char >= rg[1] and item.char <= rg[2] then
5971             Babel.chr_to_loc[item.char] = lc
5972             toloc = lc
5973             break
5974           end
5975         end
5976       end
5977       % Treat composite chars in a different fashion, because they
5978       % 'inherit' the previous locale.
5979       if (item.char >= 0x0300 and item.char <= 0x036F) or
5980         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5981         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5982         Babel.chr_to_loc[item.char] = -2000
5983         toloc = -2000
5984       end
5985       if not toloc then
5986         Babel.chr_to_loc[item.char] = -1000
5987       end
5988     end
5989     if toloc == -2000 then
5990       toloc = toloc_save
5991     elseif toloc == -1000 then
5992       toloc = nil
5993     end
5994     if toloc and Babel.locale_props[toloc] and
5995       Babel.locale_props[toloc].letters and
5996       tex.getcatcode(item.char) \string~= ll then
5997       toloc = nil
5998     end
5999     if toloc and Babel.locale_props[toloc].script
6000       and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6001       and Babel.locale_props[toloc].script ==
6002       Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6003       toloc = nil
6004     end
6005     if toloc then
6006       if Babel.locale_props[toloc].lg then
6007         item.lang = Babel.locale_props[toloc].lg
6008         node.set_attribute(item, LOCALE, toloc)
6009       end
6010       if Babel.locale_props[toloc]['/'..item.font] then
6011         item.font = Babel.locale_props[toloc]['/'..item.font]
6012       end
6013     end
6014     toloc_save = toloc
6015   elseif not inmath and item.id == 7 then % Apply recursively
6016     item.replace = item.replace and Babel.locale_map(item.replace)
6017     item.pre     = item.pre and Babel.locale_map(item.pre)
6018     item.post    = item.post and Babel.locale_map(item.post)
6019   elseif item.id == node.id'math' then
6020     inmath = (item.subtype == 0)
6021   end

```

```

6022 end
6023 return head
6024 end
6025 }

```

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

```

6026 \newcommand\babelcharproperty[1]{%
6027   \count@=#1\relax
6028   \ifvmode
6029     \expandafter\bb@chprop
6030   \else
6031     \bb@error{\string\babelcharproperty\space can be used only in\\%
6032               vertical mode (preamble or between paragraphs)}%
6033             {See the manual for further info}%
6034   \fi}
6035 \newcommand\bb@chprop[3][\the\count@]{%
6036   \@tempcnta=#1\relax
6037   \bb@ifunset{\bb@chprop@#2}%
6038   {\bb@error{No property named '#2'. Allowed values are\\%
6039             direction (bc), mirror (bmg), and linebreak (lb)}%
6040    {See the manual for further info}}%
6041   }%
6042   \loop
6043     \bb@cs{chprop@#2}{#3}%
6044     \ifnum\count@<\@tempcnta
6045       \advance\count@\@ne
6046     \repeat}
6047 \def\bb@chprop@direction#1{%
6048   \directlua{
6049     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6050     Babel.characters[\the\count@]['d'] = '#1'
6051   }}
6052 \let\bb@chprop@bc\bb@chprop@direction
6053 \def\bb@chprop@mirror#1{%
6054   \directlua{
6055     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6056     Babel.characters[\the\count@]['m'] = '\number#1'
6057   }}
6058 \let\bb@chprop@bmg\bb@chprop@mirror
6059 \def\bb@chprop@linebreak#1{%
6060   \directlua{
6061     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6062     Babel.cjk_characters[\the\count@]['c'] = '#1'
6063   }}
6064 \let\bb@chprop@lb\bb@chprop@linebreak
6065 \def\bb@chprop@locale#1{%
6066   \directlua{
6067     Babel.chr_to_loc = Babel.chr_to_loc or {}
6068     Babel.chr_to_loc[\the\count@] =
6069     \bb@ifblank{#1}{-1000}{\the\bb@cs{id@#1}}\space
6070   }}

```

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.

```

6071 \directlua{
6072   Babel.nohyphenation = \the\l@nohyphenation
6073 }

```

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, `pre={1}{1}-` becomes `function(m) return m[1]..m[1]..'-' end`, where `m` are the matches returned after applying the pattern. With a mapped capture the functions are similar to `function(m) return Babel.capt_map(m[1],1) end`, where the last argument identifies the

mapping to be applied to `m[1]`. The way it is carried out is somewhat tricky, but the effect is not dissimilar to `lua load` – save the code as string in a TeX macro, and expand this macro at the appropriate place. As `\directlua` does not take into account the current catcode of `@`, we just avoid this character in macro names (which explains the internal group, too).

```

6074 \begingroup
6075 \catcode`\~ = 12
6076 \catcode`\% = 12
6077 \catcode`\& = 14
6078 \catcode`\| = 12
6079 \gdef\babelprehyphenation{%&
6080 \ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}]
6081 \gdef\babelposthyphenation{%&
6082 \ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}]
6083 \gdef\bbl@settransform#1[#2]#3#4#5{%&
6084 \ifcase#1
6085 \bbl@activateprehyphen
6086 \or
6087 \bbl@activateposthyphen
6088 \fi
6089 \begingroup
6090 \def\babeltempa{\bbl@add@list\babeltempb}%&
6091 \let\babeltempb@empty
6092 \def\bbl@tempa{#5}%&
6093 \bbl@replace\bbl@tempa{,}{,}%& TODO. Ugly trick to preserve {}
6094 \expandafter\bbl@foreach\expandafter{\bbl@tempa}{%&
6095 \bbl@ifsamestring{##1}{remove}%&
6096 {\bbl@add@list\babeltempb{nil}}}%&
6097 {\directlua{
6098 local rep = [= [##1] =]
6099 rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6100 rep = rep:gsub('^%s*(insert)%s*', ', 'insert = true, ')
6101 rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
6102 if #1 == 0 or #1 == 2 then
6103 rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
6104 'space = {' .. '%2, %3, %4' .. '}')
6105 rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
6106 'spacefactor = {' .. '%2, %3, %4' .. '}')
6107 rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
6108 else
6109 rep = rep:gsub('(no)%s*=%s*([^\s,]*)', Babel.capture_func)
6110 rep = rep:gsub('(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
6111 rep = rep:gsub('(post)%s*=%s*([^\s,]*)', Babel.capture_func)
6112 end
6113 tex.print([[string\babeltempa{[]} .. rep .. [{}]])
6114 ]}}}%&
6115 \bbl@foreach\babeltempb{%&
6116 \bbl@forkv{##1}{%&
6117 \in{,###1,}{,nil,step,data,remove,insert,string,no,pre,&
6118 no,post,penalty,kashida,space,spacefactor,}%&
6119 \ifin@else
6120 \bbl@error
6121 {Bad option '###1' in a transform.\\&
6122 I'll ignore it but expect more errors}%&
6123 {See the manual for further info.}%&
6124 \fi}}}%&
6125 \let\bbl@kv@attribute\relax
6126 \let\bbl@kv@label\relax
6127 \let\bbl@kv@fonts@empty
6128 \bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}}%&
6129 \ifx\bbl@kv@fonts@empty\else\bbl@settransform\fi
6130 \ifx\bbl@kv@attribute\relax
6131 \ifx\bbl@kv@label\relax\else
6132 \bbl@exp{\bbl@trim@def{\bbl@kv@fonts}{\bbl@kv@fonts}}}%&

```

```

6133 \bbl@replace\bbl@kv@fonts{ }{,}&%
6134 \edef\bbl@kv@attribute{\bbl@ATR@\bbl@kv@label @#3@\bbl@kv@fonts}&%
6135 \count@ \z@
6136 \def\bbl@elt##1##2##3{&%
6137 \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
6138 {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
6139 {\count@\@ne}&%
6140 {\bbl@error
6141 {Transforms cannot be re-assigned to different\\&%
6142 fonts. The conflict is in '\bbl@kv@label'.\\&%
6143 Apply the same fonts or use a different label}&%
6144 {See the manual for further details.}}&%
6145 {}}&%
6146 \bbl@transfont@list
6147 \ifnum\count@=\z@
6148 \bbl@exp{\global\\ \bbl@add\\ \bbl@transfont@list
6149 {\ \bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
6150 \fi
6151 \bbl@ifunset{\bbl@kv@attribute}&%
6152 {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6153 {}&%
6154 \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
6155 \fi
6156 \else
6157 \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6158 \fi
6159 \directlua{
6160 local lbrk = Babel.linebreaking.replacements[#1]
6161 local u = unicode.utf8
6162 local id, attr, label
6163 if #1 == 0 then
6164 id = \the\csname bbl@id@#3\endcsname\space
6165 else
6166 id = \the\csname l@#3\endcsname\space
6167 end
6168 \ifx\bbl@kv@attribute\relax
6169 attr = -1
6170 \else
6171 attr = luatexbase.registernumber'\bbl@kv@attribute'
6172 \fi
6173 \ifx\bbl@kv@label\relax\else &% Same refs:
6174 label = [==[\bbl@kv@label]==]
6175 \fi
6176 &% Convert pattern:
6177 local patt = string.gsub([==[#4]==], '%s', '')
6178 if #1 == 0 then
6179 patt = string.gsub(patt, '|', ' ')
6180 end
6181 if not u.find(patt, '()', nil, true) then
6182 patt = '()' .. patt .. '()'
6183 end
6184 if #1 == 1 then
6185 patt = string.gsub(patt, '%(%)%^', '^()')
6186 patt = string.gsub(patt, '%$%(%)', '()$')
6187 end
6188 patt = u.gsub(patt, '{(.)}',
6189 function (n)
6190 return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6191 end)
6192 patt = u.gsub(patt, '{(%x%x%x%x+)}',
6193 function (n)
6194 return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6195 end)

```

```

6196     lbkr[id] = lbkr[id] or {}
6197     table.insert(lbkr[id],
6198       { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6199   }&%
6200 \endgroup}
6201 \endgroup
6202 \let\bbl@transfont@list@empty
6203 \def\bbl@settransfont{%
6204   \global\let\bbl@settransfont\relax % Execute only once
6205   \gdef\bbl@transfont{%
6206     \def\bbl@elt#####1#####2#####3{%
6207       \bbl@ifblank{#####3}%
6208         {\count@tw@}% Do nothing if no fonts
6209         {\count@z@
6210           \bbl@vforeach{#####3}{%
6211             \def\bbl@tempd{#####1}%
6212             \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6213             \ifx\bbl@tempd\bbl@tempe
6214               \count@\@ne
6215             \else\ifx\bbl@tempd\bbl@transfam
6216               \count@\@ne
6217             \fi\fi}%
6218             \ifcase\count@
6219               \bbl@csarg\unsetattribute{ATR@#####2@#####1@#####3}%
6220             \or
6221               \bbl@csarg\setattribute{ATR@#####2@#####1@#####3}\@ne
6222             \fi}}%
6223     \bbl@transfont@list}%
6224 \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6225 \gdef\bbl@transfam{-unknown-}%
6226 \bbl@foreach\bbl@font@fams{%
6227   \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6228   \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6229   {\xdef\bbl@transfam{##1}}%
6230   {}}}
6231 \DeclareRobustCommand\enablelocaletransform[1]{%
6232   \bbl@ifunset{\bbl@ATR@#1@\languagename @}%
6233   {\bbl@error
6234     {'#1' for '\languagename' cannot be enabled.\\%
6235     Maybe there is a typo or it's a font-dependent transform}%
6236     {See the manual for further details.}}%
6237   {\bbl@csarg\setattribute{ATR@#1@\languagename @}\@ne}}
6238 \DeclareRobustCommand\disablelocaletransform[1]{%
6239   \bbl@ifunset{\bbl@ATR@#1@\languagename @}%
6240   {\bbl@error
6241     {'#1' for '\languagename' cannot be disabled.\\%
6242     Maybe there is a typo or it's a font-dependent transform}%
6243     {See the manual for further details.}}%
6244   {\bbl@csarg\unsetattribute{ATR@#1@\languagename @}}}
6245 \def\bbl@activateposthyphen{%
6246   \let\bbl@activateposthyphen\relax
6247   \directlua{
6248     require('babel-transforms.lua')
6249     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6250   }}
6251 \def\bbl@activateprehyphen{%
6252   \let\bbl@activateprehyphen\relax
6253   \directlua{
6254     require('babel-transforms.lua')
6255     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6256   }}

```

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.

```
6257 \newcommand\localeprehyphenation[1]{%
6258 \directlua{ Babel.string_prehyphenation([=#1]==], \the\localeid) }}
```

10.9 Bidi

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

```
6259 \def\bb@activate@preotf{%
6260 \let\bb@activate@preotf\relax % only once
6261 \directlua{
6262   Babel = Babel or {}
6263   %
6264   function Babel.pre_otfload_v(head)
6265     if Babel.numbers and Babel.digits_mapped then
6266       head = Babel.numbers(head)
6267     end
6268     if Babel.bidi_enabled then
6269       head = Babel.bidi(head, false, dir)
6270     end
6271     return head
6272   end
6273   %
6274   function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6275     if Babel.numbers and Babel.digits_mapped then
6276       head = Babel.numbers(head)
6277     end
6278     if Babel.bidi_enabled then
6279       head = Babel.bidi(head, false, dir)
6280     end
6281     return head
6282   end
6283   %
6284   luatexbase.add_to_callback('pre_linebreak_filter',
6285     Babel.pre_otfload_v,
6286     'Babel.pre_otfload_v',
6287     luatexbase.priority_in_callback('pre_linebreak_filter',
6288       'luaotfload.node_processor') or nil)
6289   %
6290   luatexbase.add_to_callback('hpack_filter',
6291     Babel.pre_otfload_h,
6292     'Babel.pre_otfload_h',
6293     luatexbase.priority_in_callback('hpack_filter',
6294       'luaotfload.node_processor') or nil)
6295 }
```

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 `\bb@mathboxdir` hack is activated every math with the package option `bidi=`.

```
6296 \breakafterdirmode=1
6297 \ifnum\bb@bidimode>\@ne % Any bidi= except default=1
6298 \let\bb@beforeforeign\leavevmode
6299 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6300 \RequirePackage{luatexbase}
6301 \bb@activate@preotf
6302 \directlua{
6303   require('babel-data-bidi.lua')
6304   \ifcase\expandafter\@gobbletwo\the\bb@bidimode\or
6305     require('babel-bidi-basic.lua')
```

```

6306 \or
6307     require('babel-bidi-basic-r.lua')
6308 \fi}
6309 \newattribute\bb@attr@dir
6310 \directlua{ Babel.attr_dir = luatexbase.registernumber'bb@attr@dir' }
6311 \bb@exp{\output{\bodydir\pagedir\the\output}}
6312 \fi
6313 \chardef\bb@thetextdir\z@
6314 \chardef\bb@thepardir\z@
6315 \def\bb@getluadir#1{%
6316 \directlua{
6317     if tex.#ldir == 'TLT' then
6318         tex.sprint('0')
6319     elseif tex.#ldir == 'TRT' then
6320         tex.sprint('1')
6321     end}}
6322 \def\bb@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6323 \ifcase#3\relax
6324     \ifcase\bb@getluadir{#1}\relax\else
6325         #2 TLT\relax
6326     \fi
6327 \else
6328     \ifcase\bb@getluadir{#1}\relax
6329         #2 TRT\relax
6330     \fi
6331 \fi}
6332 % ..00PPTT, with masks 0xC (par dir) and 0x3 (text dir)
6333 \def\bb@thedir{0}
6334 \def\bb@textdir#1{%
6335 \bb@setluadir{text}\textdir{#1}%
6336 \chardef\bb@thetextdir#1\relax
6337 \edef\bb@thedir{\the\numexpr\bb@thepardir*4+#1}%
6338 \setattribute\bb@attr@dir{\numexpr\bb@thepardir*4+#1}}
6339 \def\bb@pardir#1{% Used twice
6340 \bb@setluadir{par}\pardir{#1}%
6341 \chardef\bb@thepardir#1\relax}
6342 \def\bb@bodydir{\bb@setluadir{body}\bodydir}% Used once
6343 \def\bb@pagedir{\bb@setluadir{page}\pagedir}% Unused
6344 \def\bb@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.

6345 \ifnum\bb@bidimode>\z@ % Any bidi=
6346 \def\bb@insidemath{0}%
6347 \def\bb@everymath{\def\bb@insidemath{1}}
6348 \def\bb@everydisplay{\def\bb@insidemath{2}}
6349 \frozen@everymath\expandafter{%
6350 \expandafter\bb@everymath\the\frozen@everymath}
6351 \frozen@everydisplay\expandafter{%
6352 \expandafter\bb@everydisplay\the\frozen@everydisplay}
6353 \AtBeginDocument{
6354 \directlua{
6355     function Babel.math_box_dir(head)
6356         if not (token.get_macro('bb@insidemath') == '0') then
6357             if Babel.hlist_has_bidi(head) then
6358                 local d = node.new(node.id'dir')
6359                 d.dir = '+TRT'
6360                 node.insert_before(head, node.has_glyph(head), d)
6361                 for item in node.traverse(head) do
6362                     node.set_attribute(item,
6363                         Babel.attr_dir, token.get_macro('bb@thedir'))
6364                 end
6365             end
6366         end

```

```

6366     end
6367     return head
6368 end
6369   luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6370     "Babel.math_box_dir", 0)
6371 }}%
6372 \fi

```

10.10 Layout

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

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

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

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

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

```

6373 \bbl@trace{Redefinitions for bidi layout}
6374 %
6375 <<(*More package options)>> ≡
6376 \chardef\bbl@eqnpos\z@
6377 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6378 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\@tw@}
6379 <</More package options>>
6380 %
6381 \ifnum\bbl@bidimode>\z@ % Any bidi=
6382   \matheqdirmode\@ne % A luatex primitive
6383   \let\bbl@eqnodir\relax
6384   \def\bbl@eqdel{()}
6385   \def\bbl@eqnum{%
6386     {\normalfont\normalcolor
6387       \expandafter\@firstoftwo\bbl@eqdel
6388       \theequation
6389       \expandafter\@secondoftwo\bbl@eqdel}}
6390   \def\bbl@puteqno#1{\eqno\hbox{#1}}
6391   \def\bbl@putleqno#1{\leqno\hbox{#1}}
6392   \def\bbl@eqno@flip#1{%
6393     \ifdim\prelaxsize=-\maxdimen
6394       \eqno
6395       \hb@xt@.01pt{%
6396         \hb@xt@\displaywidth{\hss{#1}\glet\bbl@upset\@currentlabel}}\hss}%
6397   \else
6398     \leqno\hbox{#1}\glet\bbl@upset\@currentlabel}%
6399   \fi
6400   \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6401 \def\bbl@leqno@flip#1{%
6402   \ifdim\prelaxsize=-\maxdimen
6403     \leqno
6404     \hb@xt@.01pt{%
6405       \hss\hb@xt@\displaywidth{\#1\glet\bbl@upset\@currentlabel}\hss}}%
6406   \else

```

```

6407     \eqno\hbox{#1\glet\bb@upset\@currentlabel}%
6408     \fi
6409     \bb@exp{\def\\\@currentlabel{\bb@upset}}
6410 \AtBeginDocument{%
6411     \ifx\bb@noamsmath\relax\else
6412     \ifx\maketag@@@\@undefined % Normal equation, eqnarray
6413     \AddToHook{env/equation/begin}{%
6414         \ifnum\bb@thetextdir>\z@
6415         \def\bb@mathboxdir{\def\bb@insidemath{1}}%
6416         \let\@eqnum\bb@eqnum
6417         \edef\bb@eqnodir{\noexpand\bb@textdir{\the\bb@thetextdir}}%
6418         \chardef\bb@thetextdir\z@
6419         \bb@add\normalfont{\bb@eqnodir}%
6420         \ifcase\bb@eqnpos
6421             \let\bb@puteqno\bb@eqno@flip
6422         \or
6423             \let\bb@puteqno\bb@leqno@flip
6424         \fi
6425     \fi}%
6426 \ifnum\bb@eqnpos=\tw@\else
6427     \def\endequation{\bb@puteqno{\@eqnum}$$\@ignoretrue}%
6428 \fi
6429 \AddToHook{env/eqnarray/begin}{%
6430     \ifnum\bb@thetextdir>\z@
6431         \def\bb@mathboxdir{\def\bb@insidemath{1}}%
6432         \edef\bb@eqnodir{\noexpand\bb@textdir{\the\bb@thetextdir}}%
6433         \chardef\bb@thetextdir\z@
6434         \bb@add\normalfont{\bb@eqnodir}%
6435         \ifnum\bb@eqnpos=\@ne
6436             \def\@eqnum{%
6437                 \setbox\z@\hbox{\bb@eqnum}%
6438                 \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6439             \else
6440                 \let\@eqnum\bb@eqnum
6441             \fi
6442         \fi}
6443     % Hack. YA luatex bug?:
6444     \expandafter\bb@sreplace\csname] \endcsname{${$}{\eqno\kern.001pt$}$}%
6445 \else % amstex
6446     \bb@exp{% Hack to hide maybe undefined conditionals:
6447         \chardef\bb@eqnpos=0%
6448         \<iftagsleft>1<else>\<if@fleqn>2<fi>\<fi>\relax}%
6449     \ifnum\bb@eqnpos=\@ne
6450         \let\bb@ams@lap\hbox
6451     \else
6452         \let\bb@ams@lap\llap
6453     \fi
6454     \ExplSyntaxOn % Required by \bb@sreplace with \intertext@
6455     \bb@sreplace\intertext@\normalbaselines%
6456     {\normalbaselines
6457     \ifx\bb@eqnodir\relax\else\bb@pardir\@ne\bb@eqnodir\fi}%
6458     \ExplSyntaxOff
6459     \def\bb@ams@tagbox#1#2{#1{\bb@eqnodir#2}}% #1=hbox|@lap|flip
6460     \ifx\bb@ams@lap\hbox % leqno
6461         \def\bb@ams@flip#1{%
6462             \hbox to 0.01pt{\hss\hbox to\displaywidth{{#1}\hss}}}%
6463     \else % eqno
6464         \def\bb@ams@flip#1{%
6465             \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}\hss}}}%
6466     \fi
6467     \def\bb@ams@preset#1{%
6468         \def\bb@mathboxdir{\def\bb@insidemath{1}}%
6469         \ifnum\bb@thetextdir>\z@

```

```

6470     \edef\bbledqnodir{\noexpand\bbledtextdir{\the\bbledthetextdir}}%
6471     \bbledsreplace\textdef{\hbox}{\bbledams@tagbox\hbox}%
6472     \bbledsreplace\maketag@@{\hbox}{\bbledams@tagbox#1}%
6473     \fi}%
6474 \ifnum\bbledqnpos=\tw@%
6475     \def\bbledams@equation{%
6476     \def\bbledmathboxdir{\def\bbledinsidemath{1}}%
6477     \ifnum\bbledthetextdir>\z@
6478     \edef\bbledqnodir{\noexpand\bbledtextdir{\the\bbledthetextdir}}%
6479     \chardef\bbledthetextdir\z@
6480     \bbledadd\normalfont{\bbledeqnodir}%
6481     \ifcase\bbledqnpos
6482     \def\veqno##1##2{\bbledeqnoflip{##1##2}}%
6483     \or
6484     \def\veqno##1##2{\bbledleqnoflip{##1##2}}%
6485     \fi
6486     \fi}%
6487     \AddToHook{env/equation/begin}{\bbledams@equation}%
6488     \AddToHook{env/equation*/begin}{\bbledams@equation}%
6489     \fi
6490     \AddToHook{env/cases/begin}{\bbledams@preset\bbledams@lap}%
6491     \AddToHook{env/multline/begin}{\bbledams@preset\hbox}%
6492     \AddToHook{env/gather/begin}{\bbledams@preset\bbledams@lap}%
6493     \AddToHook{env/gather*/begin}{\bbledams@preset\bbledams@lap}%
6494     \AddToHook{env/align/begin}{\bbledams@preset\bbledams@lap}%
6495     \AddToHook{env/align*/begin}{\bbledams@preset\bbledams@lap}%
6496     \AddToHook{env/alignat/begin}{\bbledams@preset\bbledams@lap}%
6497     \AddToHook{env/alignat*/begin}{\bbledams@preset\bbledams@lap}%
6498     \AddToHook{env/eqnalign/begin}{\bbledams@preset\hbox}%
6499     % Hackish, for proper alignment. Don't ask me why it works!:
6500     \bbledexp% Avoid a 'visible' conditional
6501     \\\AddToHook{env/align*/end}{\<iftag>\<else>\\tag*{\<fi>}}%
6502     \\\AddToHook{env/alignat*/end}{\<iftag>\<else>\\tag*{\<fi>}}%
6503     \AddToHook{env/flalign/begin}{\bbledams@preset\hbox}%
6504     \AddToHook{env/split/before}{%
6505     \def\bbledmathboxdir{\def\bbledinsidemath{1}}%
6506     \ifnum\bbledthetextdir>\z@
6507     \bbledifsamestring@currentenv{equation}%
6508     {\ifx\bbledams@lap\hbox % leqno
6509     \def\bbledams@flip#1{%
6510     \hbox to 0.01pt{\hbox to\displaywidth{##1}\hss}\hss}}%
6511     \else
6512     \def\bbledams@flip#1{%
6513     \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{##1}}}}%
6514     \fi}%
6515     }%
6516     \fi}%
6517     \fi\fi}
6518 \fi
6519 \def\bbledprovide@extra#1{%
6520 % == Counters: mapdigits ==
6521 % Native digits
6522 \ifx\bbledKVP@mapdigits\@nnil\else
6523 \bbled@ifunset{bbleddgnat@languagename}{%
6524 {\RequirePackage{luatexbase}}%
6525 \bbled@activate@preotf
6526 \directlua{
6527     Babel = Babel or {} %% -> presets in luababel
6528     Babel.digits_mapped = true
6529     Babel.digits = Babel.digits or {}
6530     Babel.digits[\the\localeid] =
6531     table.pack(string.utfvalue('\bbledcl{dgnat}'))
6532     if not Babel.numbers then

```

```

6533         function Babel.numbers(head)
6534             local LOCALE = Babel.attr_locale
6535             local GLYPH = node.id'glyph'
6536             local inmath = false
6537             for item in node.traverse(head) do
6538                 if not inmath and item.id == GLYPH then
6539                     local temp = node.get_attribute(item, LOCALE)
6540                     if Babel.digits[temp] then
6541                         local chr = item.char
6542                         if chr > 47 and chr < 58 then
6543                             item.char = Babel.digits[temp][chr-47]
6544                         end
6545                     end
6546                 elseif item.id == node.id'math' then
6547                     inmath = (item.subtype == 0)
6548                 end
6549             end
6550             return head
6551         end
6552     end
6553 }}%
6554 \fi
6555 % == transforms ==
6556 \ifx\bbk@KVP@transforms\@nnil\else
6557     \def\bbk@elt##1##2##3{%
6558         \in@{transforms.}{##1}%
6559         \ifin@
6560             \def\bbk@tempa{##1}%
6561             \bbk@replace\bbk@tempa{transforms.}{}%
6562             \bbk@carg\bbk@transforms{babel\bbk@tempa}{##2}{##3}%
6563         \fi}%
6564     \csname bbl@inidata@\languagenamendcsname
6565     \bbk@release@transforms\relax % \relax closes the last item.
6566 \fi}
6567 % Start tabular here:
6568 \def\localerestoredirs{%
6569     \ifcase\bbk@thetextdir
6570         \ifnum\textdirection=\z@\else\textdir TLT\fi
6571     \else
6572         \ifnum\textdirection=\@ne\else\textdir TRT\fi
6573     \fi
6574     \ifcase\bbk@thepardir
6575         \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6576     \else
6577         \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6578     \fi}
6579 \IfBabelLayout{tabular}%
6580 {\chardef\bbk@tabular@mode\tw}% All RTL
6581 {\IfBabelLayout{notabular}%
6582     {\chardef\bbk@tabular@mode\z}%
6583     {\chardef\bbk@tabular@mode\@ne}} Mixed, with LTR cols
6584 \ifnum\bbk@bidimode>\@ne % Any lua bidi= except default=1
6585 \ifcase\bbk@tabular@mode\or % 1
6586     \let\bbk@parabefore\relax
6587     \AddToHook{para/before}{\bbk@parabefore}
6588     \AtBeginDocument{%
6589         \bbk@replace\@tabular{${}$}%
6590         \def\bbk@insidemath{0}%
6591         \def\bbk@parabefore{\localerestoredirs}}%
6592     \ifnum\bbk@tabular@mode=\@ne
6593         \bbk@ifunset{\@tabclassz}{%
6594             \bbk@exp% Hide conditionals
6595             \\bbk@sreplace\\ \@tabclassz

```

```

6596         {\<ifcase>\\@chnum}%
6597         {\localerestoredirs\<ifcase>\\@chnum}}}%
6598     \@ifpackageloaded{colortbl}%
6599         {\bbl@sreplace@classz
6600         {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}}%
6601     {\@ifpackageloaded{array}%
6602         {\bbl@exp{% Hide conditionals
6603             \\bbl@sreplace\\@classz
6604             {\<ifcase>\\@chnum}%
6605             {\bgroup\\localerestoredirs\<ifcase>\\@chnum}%
6606             \\bbl@sreplace\\@classz
6607             {\do@row@strut\<fi>}{\do@row@strut\<fi>\egroup}}}%
6608         {}}}%
6609     \fi}%
6610 \or % 2
6611     \let\bbl@parabefore\relax
6612     \AddToHook{para/before}{\bbl@parabefore}%
6613     \AtBeginDocument{%
6614         \@ifpackageloaded{colortbl}%
6615             {\bbl@replace@tabular{${}$%
6616                 \def\bbl@insidemath{0}%
6617                 \def\bbl@parabefore{\localerestoredirs}}}%
6618             \bbl@sreplace@classz
6619             {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}}%
6620         {}}}%
6621     \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.

```

6622     \AtBeginDocument{%
6623         \@ifpackageloaded{multicol}%
6624             {\toks@\expandafter{\multi@column@out}%
6625             \edef\multi@column@out{\bodydir\pagedir\the\toks@}}}%
6626         {}}%
6627     \@ifpackageloaded{paracol}%
6628         {\edef\pcol@output{%
6629             \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
6630         {}}}%
6631     \fi
6632     \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout

```

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

```

6633     \ifnum\bbl@bidimode>\z@ % Any bidi=
6634     \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6635         \bbl@exp{%
6636             \def\\bbl@insidemath{0}%
6637             \mathdir\the\bodydir
6638             #1%           Once entered in math, set boxes to restore values
6639             \<ifmmode>%
6640                 \everyvbox{%
6641                     \the\everyvbox
6642                     \bodydir\the\bodydir
6643                     \mathdir\the\mathdir
6644                     \everyhbox{\the\everyhbox}%
6645                     \everyvbox{\the\everyvbox}}}%
6646                 \everyhbox{%
6647                     \the\everyhbox
6648                     \bodydir\the\bodydir
6649                     \mathdir\the\mathdir
6650                     \everyhbox{\the\everyhbox}}%

```

```

6651     \everyvbox{\the\everyvbox}}%
6652     \<fi>}}%
6653 \def\@hangfrom#1{%
6654     \setbox\@tempboxa\hbox{#{#1}}%
6655     \hangindent\wd\@tempboxa
6656     \ifnum\bb@getluadir{page}=\bb@getluadir{par}\else
6657         \shapemode\@ne
6658     \fi
6659     \noindent\box\@tempboxa}
6660 \fi
6661 \IfBabelLayout{tabular}
6662 {\let\bb@OL@tabular\@tabular
6663  \bb@replace\@tabular{$}\{bb@nextfake$}%
6664  \let\bb@NL@tabular\@tabular
6665  \AtBeginDocument{%
6666      \ifx\bb@NL@tabular\@tabular\else
6667          \bb@exp{\in@{\bb@nextfake}\{[@tabular]}}%
6668          \ifin\else
6669              \bb@replace\@tabular{$}\{bb@nextfake$}%
6670          \fi
6671          \let\bb@NL@tabular\@tabular
6672      \fi}}
6673 {}
6674 \IfBabelLayout{lists}
6675 {\let\bb@OL@list\list
6676  \bb@sreplace\list{\parshape}\{bb@listparshape}%
6677  \let\bb@NL@list\list
6678  \def\bb@listparshape#1#2#3{%
6679      \parshape #1 #2 #3 %
6680      \ifnum\bb@getluadir{page}=\bb@getluadir{par}\else
6681          \shapemode\tw@
6682      \fi}}
6683 {}
6684 \IfBabelLayout{graphics}
6685 {\let\bb@pictresetdir\relax
6686  \def\bb@pictsetdir#1{%
6687      \ifcase\bb@thetextdir
6688          \let\bb@pictresetdir\relax
6689      \else
6690          \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6691              \or\textdir TLT
6692              \else\bodydir TLT \textdir TLT
6693          \fi
6694          % \(\text|par)dir required in pgf:
6695          \def\bb@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6696      \fi}%
6697  \AddToHook{env/picture/begin}{\bb@pictsetdir\tw@}%
6698  \directlua{
6699      Babel.get_picture_dir = true
6700      Babel.picture_has_bidi = 0
6701      %
6702      function Babel.picture_dir (head)
6703          if not Babel.get_picture_dir then return head end
6704          if Babel.hlist_has_bidi(head) then
6705              Babel.picture_has_bidi = 1
6706          end
6707          return head
6708      end
6709      luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6710      "Babel.picture_dir")
6711  }%
6712  \AtBeginDocument{%
6713      \def\LS@rot{%

```



```

6714     \setbox\@outputbox\vbox{%
6715     \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
6716 \long\def\put(#1,#2)#3{%
6717     \@killglove
6718     % Try:
6719     \ifx\bbbl@pictresetdir\relax
6720     \def\bbbl@tempc{0}%
6721     \else
6722     \directlua{
6723     Babel.get_picture_dir = true
6724     Babel.picture_has_bidi = 0
6725     }%
6726     \setbox\z@\hb@xt@\z@{%
6727     \@defaultunitsset\@tempdimc{#1}\unitlength
6728     \kern\@tempdimc
6729     #3\hss}% TODO: #3 executed twice (below). That's bad.
6730     \edef\bbbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6731     \fi
6732     % Do:
6733     \@defaultunitsset\@tempdimc{#2}\unitlength
6734     \raise\@tempdimc\hb@xt@\z@{%
6735     \@defaultunitsset\@tempdimc{#1}\unitlength
6736     \kern\@tempdimc
6737     {\ifnum\bbbl@tempc>\z@\bbbl@pictresetdir\fi#3}\hss}%
6738     \ignorespaces}%
6739     \MakeRobust\put}%
6740 \AtBeginDocument
6741 {\AddToHook{cmd/diagbox@pict/before}{\let\bbbl@pictsetdir\@gobble}%
6742 \ifx\pgfpicture\@undefined\else % TODO. Allow deactivate?
6743 \AddToHook{env/pgfpicture/begin}{\bbbl@pictsetdir\@ne}%
6744 \bbbl@add\pgfinterruptpicture{\bbbl@pictresetdir}%
6745 \bbbl@add\pgfsys@beginpicture{\bbbl@pictsetdir\z@}%
6746 \fi
6747 \ifx\tikzpicture\@undefined\else
6748 \AddToHook{env/tikzpicture/begin}{\bbbl@pictsetdir\tw@}%
6749 \bbbl@add\tikz@atbegin@node{\bbbl@pictresetdir}%
6750 \bbbl@sreplace\tikz{\begingroup}{\begingroup\bbbl@pictsetdir\tw@}%
6751 \fi
6752 \ifx\tcolorbox\@undefined\else
6753 \def\tcb@drawing@env@begin{%
6754 \csname tcb@before@tcb@split@state\endcsname
6755 \bbbl@pictsetdir\tw@
6756 \begin{\kvtcb@graphenv}%
6757 \tcb@bbdraw%
6758 \tcb@apply@graph@patches
6759 }%
6760 \def\tcb@drawing@env@end{%
6761 \end{\kvtcb@graphenv}%
6762 \bbbl@pictresetdir
6763 \csname tcb@after@tcb@split@state\endcsname
6764 }%
6765 \fi
6766 }}
6767 {}

```

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.

```

6768 \IfBabelLayout{counters*}%
6769 {\bbbl@add\bbbl@opt@layout{.counters.}%
6770 \directlua{
6771     luatexbase.add_to_callback("process_output_buffer",
6772     Babel.discard_sublr , "Babel.discard_sublr") }%

```

```

6773   {}
6774 \IfBabelLayout{counters}%
6775   {\let\bbL@0L@textsuperscript\@textsuperscript
6776    \bbL@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6777    \let\bbL@latinarabic=\@arabic
6778    \let\bbL@0L@arabic\@arabic
6779    \def\@arabic#1{\babelsublr{\bbL@latinarabic#1}}%
6780    \@ifpackagewith{babel}{bidi=default}%
6781     {\let\bbL@asciroman=\@roman
6782      \let\bbL@0L@roman\@roman
6783      \def\@roman#1{\babelsublr{\ensureascii{\bbL@asciroman#1}}}%
6784      \let\bbL@asciiRoman=\@Roman
6785      \let\bbL@0L@roman\@Roman
6786      \def\@Roman#1{\babelsublr{\ensureascii{\bbL@asciiRoman#1}}}%
6787      \let\bbL@0L@labelenumii\labelenumii
6788      \def\labelenumii{}\theenumii}%
6789      \let\bbL@0L@p@enumiii\p@enumiii
6790      \def\p@enumiii{\p@enumii}\theenumii{}}{}
6791 \langle Footnote changes \rangle
6792 \IfBabelLayout{footnotes}%
6793   {\let\bbL@0L@footnote\footnote
6794    \BabelFootnote\footnote\languagename{}}{}%
6795    \BabelFootnote\localfootnote\languagename{}}{}%
6796    \BabelFootnote\mainfootnote{}}{}
6797   {}

```

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

```

6798 \IfBabelLayout{extras}%
6799   {\bbL@ncarg\let\bbL@0L@underline{underline }%
6800    \bbL@carg\bbL@sreplace{underline }%
6801     {\$@@underline}{\bgroup\bbL@nextfake$@@underline}%
6802    \bbL@carg\bbL@sreplace{underline }%
6803     {\m@th$}{\m@th$\egroup}%
6804    \let\bbL@0L@LaTeXe\LaTeXe
6805    \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6806     \if b\expandafter\@car\@f@series\@nil\boldmath\fi
6807     \babelsublr{%
6808      \LaTeX\kern.15em2\bbL@nextfake$_{\textstyle\varepsilon}$}}}
6809   {}
6810 \langle luatex \rangle

```

10.11 Lua: transforms

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

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

```

6811 \langle *transforms \rangle
6812 Babel.linebreaking.replacements = {}
6813 Babel.linebreaking.replacements[0] = {} -- pre
6814 Babel.linebreaking.replacements[1] = {} -- post
6815
6816 -- Discretionaries contain strings as nodes
6817 function Babel.str_to_nodes(fn, matches, base)
6818   local n, head, last

```

```

6819 if fn == nil then return nil end
6820 for s in string.utfvalues(fn(matches)) do
6821   if base.id == 7 then
6822     base = base.replace
6823   end
6824   n = node.copy(base)
6825   n.char = s
6826   if not head then
6827     head = n
6828   else
6829     last.next = n
6830   end
6831   last = n
6832 end
6833 return head
6834 end
6835
6836 Babel.fetch_subtext = {}
6837
6838 Babel.ignore_pre_char = function(node)
6839   return (node.lang == Babel.nohyphenation)
6840 end
6841
6842 -- Merging both functions doesn't seem feasible, because there are too
6843 -- many differences.
6844 Babel.fetch_subtext[0] = function(head)
6845   local word_string = ''
6846   local word_nodes = {}
6847   local lang
6848   local item = head
6849   local inmath = false
6850
6851   while item do
6852
6853     if item.id == 11 then
6854       inmath = (item.subtype == 0)
6855     end
6856
6857     if inmath then
6858       -- pass
6859
6860     elseif item.id == 29 then
6861       local locale = node.get_attribute(item, Babel.attr_locale)
6862
6863       if lang == locale or lang == nil then
6864         lang = lang or locale
6865         if Babel.ignore_pre_char(item) then
6866           word_string = word_string .. Babel.us_char
6867         else
6868           word_string = word_string .. unicode.utf8.char(item.char)
6869         end
6870         word_nodes[#word_nodes+1] = item
6871       else
6872         break
6873       end
6874
6875     elseif item.id == 12 and item.subtype == 13 then
6876       word_string = word_string .. ' '
6877       word_nodes[#word_nodes+1] = item
6878
6879     -- Ignore leading unrecognized nodes, too.
6880     elseif word_string ~= '' then
6881       word_string = word_string .. Babel.us_char

```

```

6882     word_nodes[#word_nodes+1] = item -- Will be ignored
6883 end
6884
6885     item = item.next
6886 end
6887
6888 -- Here and above we remove some trailing chars but not the
6889 -- corresponding nodes. But they aren't accessed.
6890 if word_string:sub(-1) == ' ' then
6891     word_string = word_string:sub(1,-2)
6892 end
6893 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6894 return word_string, word_nodes, item, lang
6895 end
6896
6897 Babel.fetch_subtext[1] = function(head)
6898     local word_string = ''
6899     local word_nodes = {}
6900     local lang
6901     local item = head
6902     local inmath = false
6903
6904     while item do
6905
6906         if item.id == 11 then
6907             inmath = (item.subtype == 0)
6908         end
6909
6910         if inmath then
6911             -- pass
6912
6913         elseif item.id == 29 then
6914             if item.lang == lang or lang == nil then
6915                 if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6916                     lang = lang or item.lang
6917                     word_string = word_string .. unicode.utf8.char(item.char)
6918                     word_nodes[#word_nodes+1] = item
6919                 end
6920             else
6921                 break
6922             end
6923
6924         elseif item.id == 7 and item.subtype == 2 then
6925             word_string = word_string .. '='
6926             word_nodes[#word_nodes+1] = item
6927
6928         elseif item.id == 7 and item.subtype == 3 then
6929             word_string = word_string .. '|'
6930             word_nodes[#word_nodes+1] = item
6931
6932         -- (1) Go to next word if nothing was found, and (2) implicitly
6933         -- remove leading USs.
6934         elseif word_string == '' then
6935             -- pass
6936
6937         -- This is the responsible for splitting by words.
6938         elseif (item.id == 12 and item.subtype == 13) then
6939             break
6940
6941         else
6942             word_string = word_string .. Babel.us_char
6943             word_nodes[#word_nodes+1] = item -- Will be ignored
6944         end

```

```

6945
6946     item = item.next
6947 end
6948
6949 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6950 return word_string, word_nodes, item, lang
6951 end
6952
6953 function Babel.pre_hyphenate_replace(head)
6954     Babel.hyphenate_replace(head, 0)
6955 end
6956
6957 function Babel.post_hyphenate_replace(head)
6958     Babel.hyphenate_replace(head, 1)
6959 end
6960
6961 Babel.us_char = string.char(31)
6962
6963 function Babel.hyphenate_replace(head, mode)
6964     local u = unicode.utf8
6965     local lbkr = Babel.linebreaking.replacements[mode]
6966
6967     local word_head = head
6968
6969     while true do -- for each subtext block
6970
6971         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6972
6973         if Babel.debug then
6974             print()
6975             print((mode == 0) and '@@@@<' or '@@@@>', w)
6976         end
6977
6978         if nw == nil and w == '' then break end
6979
6980         if not lang then goto next end
6981         if not lbkr[lang] then goto next end
6982
6983         -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6984         -- loops are nested.
6985         for k=1, #lbkr[lang] do
6986             local p = lbkr[lang][k].pattern
6987             local r = lbkr[lang][k].replace
6988             local attr = lbkr[lang][k].attr or -1
6989
6990             if Babel.debug then
6991                 print('*****', p, mode)
6992             end
6993
6994             -- This variable is set in some cases below to the first *byte*
6995             -- after the match, either as found by u.match (faster) or the
6996             -- computed position based on sc if w has changed.
6997             local last_match = 0
6998             local step = 0
6999
7000             -- For every match.
7001             while true do
7002                 if Babel.debug then
7003                     print('====')
7004                 end
7005                 local new -- used when inserting and removing nodes
7006
7007                 local matches = { u.match(w, p, last_match) }

```

```

7008
7009     if #matches < 2 then break end
7010
7011     -- Get and remove empty captures (with ())'s, which return a
7012     -- number with the position), and keep actual captures
7013     -- (from (...)), if any, in matches.
7014     local first = table.remove(matches, 1)
7015     local last  = table.remove(matches, #matches)
7016     -- Non re-fetched substrings may contain \31, which separates
7017     -- subsubstrings.
7018     if string.find(w:sub(first, last-1), Babel.us_char) then break end
7019
7020     local save_last = last -- with A()BC()D, points to D
7021
7022     -- Fix offsets, from bytes to unicode. Explained above.
7023     first = u.len(w:sub(1, first-1)) + 1
7024     last  = u.len(w:sub(1, last-1)) -- now last points to C
7025
7026     -- This loop stores in a small table the nodes
7027     -- corresponding to the pattern. Used by 'data' to provide a
7028     -- predictable behavior with 'insert' (w_nodes is modified on
7029     -- the fly), and also access to 'remove'd nodes.
7030     local sc = first-1      -- Used below, too
7031     local data_nodes = {}
7032
7033     local enabled = true
7034     for q = 1, last-first+1 do
7035         data_nodes[q] = w_nodes[sc+q]
7036         if enabled
7037             and attr > -1
7038             and not node.has_attribute(data_nodes[q], attr)
7039         then
7040             enabled = false
7041         end
7042     end
7043
7044     -- This loop traverses the matched substring and takes the
7045     -- corresponding action stored in the replacement list.
7046     -- sc = the position in substr nodes / string
7047     -- rc = the replacement table index
7048     local rc = 0
7049
7050     while rc < last-first+1 do -- for each replacement
7051         if Babel.debug then
7052             print('.....', rc + 1)
7053         end
7054         sc = sc + 1
7055         rc = rc + 1
7056
7057         if Babel.debug then
7058             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7059             local ss = ''
7060             for itt in node.traverse(head) do
7061                 if itt.id == 29 then
7062                     ss = ss .. unicode.utf8.char(itt.char)
7063                 else
7064                     ss = ss .. '{' .. itt.id .. '}'
7065                 end
7066             end
7067             print('*****', ss)
7068         end
7069     end
7070

```

```

7071     local crep = r[rc]
7072     local item = w_nodes[sc]
7073     local item_base = item
7074     local placeholder = Babel.us_char
7075     local d
7076
7077     if crep and crep.data then
7078         item_base = data_nodes[crep.data]
7079     end
7080
7081     if crep then
7082         step = crep.step or 0
7083     end
7084
7085     if (not enabled) or (crep and next(crep) == nil) then -- = {}
7086         last_match = save_last -- Optimization
7087         goto next
7088
7089     elseif crep == nil or crep.remove then
7090         node.remove(head, item)
7091         table.remove(w_nodes, sc)
7092         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7093         sc = sc - 1 -- Nothing has been inserted.
7094         last_match = utf8.offset(w, sc+1+step)
7095         goto next
7096
7097     elseif crep and crep.kashida then -- Experimental
7098         node.set_attribute(item,
7099             Babel.attr_kashida,
7100             crep.kashida)
7101         last_match = utf8.offset(w, sc+1+step)
7102         goto next
7103
7104     elseif crep and crep.string then
7105         local str = crep.string(matches)
7106         if str == '' then -- Gather with nil
7107             node.remove(head, item)
7108             table.remove(w_nodes, sc)
7109             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7110             sc = sc - 1 -- Nothing has been inserted.
7111         else
7112             local loop_first = true
7113             for s in string.utfvalues(str) do
7114                 d = node.copy(item_base)
7115                 d.char = s
7116                 if loop_first then
7117                     loop_first = false
7118                     head, new = node.insert_before(head, item, d)
7119                     if sc == 1 then
7120                         word_head = head
7121                     end
7122                     w_nodes[sc] = d
7123                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
7124                 else
7125                     sc = sc + 1
7126                     head, new = node.insert_before(head, item, d)
7127                     table.insert(w_nodes, sc, new)
7128                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7129                 end
7130                 if Babel.debug then
7131                     print('.....', 'str')
7132                     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7133                 end
7134             end
7135         end
7136     end

```

```

7134         end -- for
7135         node.remove(head, item)
7136     end -- if ''
7137     last_match = utf8.offset(w, sc+1+step)
7138     goto next
7139
7140 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7141     d = node.new(7, 3) -- (disc, regular)
7142     d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7143     d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7144     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7145     d.attr = item_base.attr
7146     if crep.pre == nil then -- TeXbook p96
7147         d.penalty = crep.penalty or tex.hyphenpenalty
7148     else
7149         d.penalty = crep.penalty or tex.exhyphenpenalty
7150     end
7151     placeholder = '|'
7152     head, new = node.insert_before(head, item, d)
7153
7154 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7155     -- ERROR
7156
7157 elseif crep and crep.penalty then
7158     d = node.new(14, 0) -- (penalty, userpenalty)
7159     d.attr = item_base.attr
7160     d.penalty = crep.penalty
7161     head, new = node.insert_before(head, item, d)
7162
7163 elseif crep and crep.space then
7164     -- 655360 = 10 pt = 10 * 65536 sp
7165     d = node.new(12, 13) -- (glue, spaceskip)
7166     local quad = font.getfont(item_base.font).size or 655360
7167     node.setglue(d, crep.space[1] * quad,
7168                  crep.space[2] * quad,
7169                  crep.space[3] * quad)
7170     if mode == 0 then
7171         placeholder = ' '
7172     end
7173     head, new = node.insert_before(head, item, d)
7174
7175 elseif crep and crep.spacefactor then
7176     d = node.new(12, 13) -- (glue, spaceskip)
7177     local base_font = font.getfont(item_base.font)
7178     node.setglue(d,
7179                 crep.spacefactor[1] * base_font.parameters['space'],
7180                 crep.spacefactor[2] * base_font.parameters['space_stretch'],
7181                 crep.spacefactor[3] * base_font.parameters['space_shrink'])
7182     if mode == 0 then
7183         placeholder = ' '
7184     end
7185     head, new = node.insert_before(head, item, d)
7186
7187 elseif mode == 0 and crep and crep.space then
7188     -- ERROR
7189
7190 end -- ie replacement cases
7191
7192 -- Shared by disc, space and penalty.
7193 if sc == 1 then
7194     word_head = head
7195 end
7196 if crep.insert then

```



```

7197         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7198         table.insert(w_nodes, sc, new)
7199         last = last + 1
7200     else
7201         w_nodes[sc] = d
7202         node.remove(head, item)
7203         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7204     end
7205
7206     last_match = utf8.offset(w, sc+1+step)
7207
7208     ::next::
7209
7210     end -- for each replacement
7211
7212     if Babel.debug then
7213         print('.....', '/')
7214         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7215     end
7216
7217     end -- for match
7218
7219     end -- for patterns
7220
7221     ::next::
7222     word_head = nw
7223     end -- for substring
7224     return head
7225 end
7226
7227 -- This table stores capture maps, numbered consecutively
7228 Babel.capture_maps = {}
7229
7230 -- The following functions belong to the next macro
7231 function Babel.capture_func(key, cap)
7232     local ret = "[" .. cap:gsub('{{[0-9]}}', ")]..m[%1]..["] .. "]"
7233     local cnt
7234     local u = unicode.utf8
7235     ret, cnt = ret:gsub('{{[0-9]}|([^\]]+)|(.-)}', Babel.capture_func_map)
7236     if cnt == 0 then
7237         ret = u.gsub(ret, '{{(%X%X%X%X+)}',
7238             function (n)
7239                 return u.char(tonumber(n, 16))
7240             end)
7241     end
7242     ret = ret:gsub("%[%[%]]%.", '')
7243     ret = ret:gsub("%.%[%[%]]%", '')
7244     return key .. [[=function(m) return ]] .. ret .. [[ end]]
7245 end
7246
7247 function Babel.capt_map(from, mapno)
7248     return Babel.capture_maps[mapno][from] or from
7249 end
7250
7251 -- Handle the {n|abc|ABC} syntax in captures
7252 function Babel.capture_func_map(capno, from, to)
7253     local u = unicode.utf8
7254     from = u.gsub(from, '{{(%X%X%X%X+)}',
7255         function (n)
7256             return u.char(tonumber(n, 16))
7257         end)
7258     to = u.gsub(to, '{{(%X%X%X%X+)}',
7259         function (n)

```

```

7260         return u.char(tonumber(n, 16))
7261     end)
7262     local froms = {}
7263     for s in string.utfcharacters(from) do
7264         table.insert(froms, s)
7265     end
7266     local cnt = 1
7267     table.insert(Babel.capture_maps, {})
7268     local mlen = table.getn(Babel.capture_maps)
7269     for s in string.utfcharacters(to) do
7270         Babel.capture_maps[mlen][froms[cnt]] = s
7271         cnt = cnt + 1
7272     end
7273     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
7274         (mlen) .. ").." .. "[["
7275 end
7276
7277 -- Create/Extend reversed sorted list of kashida weights:
7278 function Babel.capture_kashida(key, wt)
7279     wt = tonumber(wt)
7280     if Babel.kashida_wts then
7281         for p, q in ipairs(Babel.kashida_wts) do
7282             if wt == q then
7283                 break
7284             elseif wt > q then
7285                 table.insert(Babel.kashida_wts, p, wt)
7286                 break
7287             elseif table.getn(Babel.kashida_wts) == p then
7288                 table.insert(Babel.kashida_wts, wt)
7289             end
7290         end
7291     else
7292         Babel.kashida_wts = { wt }
7293     end
7294     return 'kashida = ' .. wt
7295 end
7296
7297 -- Experimental: applies prehyphenation transforms to a string (letters
7298 -- and spaces).
7299 function Babel.string_prehyphenation(str, locale)
7300     local n, head, last, res
7301     head = node.new(8, 0) -- dummy (hack just to start)
7302     last = head
7303     for s in string.utfvalues(str) do
7304         if s == 20 then
7305             n = node.new(12, 0)
7306         else
7307             n = node.new(29, 0)
7308             n.char = s
7309         end
7310         node.set_attribute(n, Babel.attr_locale, locale)
7311         last.next = n
7312         last = n
7313     end
7314     head = Babel.hyphenate_replace(head, 0)
7315     res = ''
7316     for n in node.traverse(head) do
7317         if n.id == 12 then
7318             res = res .. ' '
7319         elseif n.id == 29 then
7320             res = res .. unicode.utf8.char(n.char)
7321         end
7322     end

```

```

7323 tex.print(res)
7324 end
7325 </transforms>

```

10.12 Lua: Auto bidi with basic and basic-r

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

```

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

```

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

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

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

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

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

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

```

7326 (*basic-r)
7327 Babel = Babel or {}
7328
7329 Babel.bidi_enabled = true
7330
7331 require('babel-data-bidi.lua')
7332
7333 local characters = Babel.characters
7334 local ranges = Babel.ranges
7335
7336 local DIR = node.id("dir")
7337
7338 local function dir_mark(head, from, to, outer)
7339   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
7340   local d = node.new(DIR)
7341   d.dir = '+' .. dir
7342   node.insert_before(head, from, d)
7343   d = node.new(DIR)
7344   d.dir = '-' .. dir
7345   node.insert_after(head, to, d)
7346 end

```

```

7347
7348 function Babel.bidi(head, ispar)
7349   local first_n, last_n          -- first and last char with nums
7350   local last_es                  -- an auxiliary 'last' used with nums
7351   local first_d, last_d         -- first and last char in L/R block
7352   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):

```

7353   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7354   local strong_lr = (strong == 'l') and 'l' or 'r'
7355   local outer = strong
7356
7357   local new_dir = false
7358   local first_dir = false
7359   local inmath = false
7360
7361   local last_lr
7362
7363   local type_n = ''
7364
7365   for item in node.traverse(head) do
7366     -- three cases: glyph, dir, otherwise
7367     if item.id == node.id'glyph'
7368       or (item.id == 7 and item.subtype == 2) then
7370
7371       local itemchar
7372       if item.id == 7 and item.subtype == 2 then
7373         itemchar = item.replace.char
7374       else
7375         itemchar = item.char
7376       end
7377       local chardata = characters[itemchar]
7378       dir = chardata and chardata.d or nil
7379       if not dir then
7380         for nn, et in ipairs(ranges) do
7381           if itemchar < et[1] then
7382             break
7383           elseif itemchar <= et[2] then
7384             dir = et[3]
7385             break
7386           end
7387         end
7388       end
7389       dir = dir or 'l'
7390       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).

```

7391     if new_dir then
7392       attr_dir = 0
7393       for at in node.traverse(item.attr) do
7394         if at.number == Babel.attr_dir then
7395           attr_dir = at.value & 0x3
7396         end
7397       end
7398       if attr_dir == 1 then
7399         strong = 'r'
7400       elseif attr_dir == 2 then

```

```

7401     strong = 'al'
7402     else
7403     strong = 'l'
7404     end
7405     strong_lr = (strong == 'l') and 'l' or 'r'
7406     outer = strong_lr
7407     new_dir = false
7408 end
7409
7410 if dir == 'nsm' then dir = strong end      -- W1

```

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

```

7411     dir_real = dir          -- We need dir_real to set strong below
7412     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:

```

7413     if strong == 'al' then
7414         if dir == 'en' then dir = 'an' end      -- W2
7415         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7416         strong_lr = 'r'                        -- W3
7417     end

```

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

```

7418     elseif item.id == node.id'dir' and not inmath then
7419         new_dir = true
7420         dir = nil
7421     elseif item.id == node.id'math' then
7422         inmath = (item.subtype == 0)
7423     else
7424         dir = nil      -- Not a char
7425     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>.

```

7426     if dir == 'en' or dir == 'an' or dir == 'et' then
7427         if dir ~= 'et' then
7428             type_n = dir
7429         end
7430         first_n = first_n or item
7431         last_n = last_es or item
7432         last_es = nil
7433     elseif dir == 'es' and last_n then -- W3+W6
7434         last_es = item
7435     elseif dir == 'cs' then          -- it's right - do nothing
7436     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7437         if strong_lr == 'r' and type_n ~= '' then
7438             dir_mark(head, first_n, last_n, 'r')
7439         elseif strong_lr == 'l' and first_d and type_n == 'an' then
7440             dir_mark(head, first_n, last_n, 'r')
7441             dir_mark(head, first_d, last_d, outer)
7442             first_d, last_d = nil, nil
7443         elseif strong_lr == 'l' and type_n ~= '' then
7444             last_d = last_n
7445         end
7446         type_n = ''
7447         first_n, last_n = nil, nil
7448     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:

```
7449   if dir == 'l' or dir == 'r' then
7450     if dir ~= outer then
7451       first_d = first_d or item
7452       last_d = item
7453     elseif first_d and dir ~= strong_lr then
7454       dir_mark(head, first_d, last_d, outer)
7455       first_d, last_d = nil, nil
7456     end
7457   end
```

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

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

```
7458   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7459     item.char = characters[item.char] and
7460       characters[item.char].m or item.char
7461   elseif (dir or new_dir) and last_lr ~= item then
7462     local mir = outer .. strong_lr .. (dir or outer)
7463     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7464       for ch in node.traverse(node.next(last_lr)) do
7465         if ch == item then break end
7466         if ch.id == node.id'glyph' and characters[ch.char] then
7467           ch.char = characters[ch.char].m or ch.char
7468         end
7469       end
7470     end
7471   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).

```
7472   if dir == 'l' or dir == 'r' then
7473     last_lr = item
7474     strong = dir_real      -- Don't search back - best save now
7475     strong_lr = (strong == 'l') and 'l' or 'r'
7476   elseif new_dir then
7477     last_lr = nil
7478   end
7479   end
```

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

```
7480   if last_lr and outer == 'r' then
7481     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7482       if characters[ch.char] then
7483         ch.char = characters[ch.char].m or ch.char
7484       end
7485     end
7486   end
7487   if first_n then
7488     dir_mark(head, first_n, last_n, outer)
7489   end
7490   if first_d then
7491     dir_mark(head, first_d, last_d, outer)
7492   end
```

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

```
7493   return node.prev(head) or head
7494 end
7495 </basic-r>
```

And here the Lua code for bidi=basic:

```
7496 (*basic)
7497 Babel = Babel or {}
7498
7499 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
7500
7501 Babel.fontmap = Babel.fontmap or {}
7502 Babel.fontmap[0] = {} -- l
7503 Babel.fontmap[1] = {} -- r
7504 Babel.fontmap[2] = {} -- al/an
7505
7506 Babel.bidi_enabled = true
7507 Babel.mirroring_enabled = true
7508
7509 require('babel-data-bidi.lua')
7510
7511 local characters = Babel.characters
7512 local ranges = Babel.ranges
7513
7514 local DIR = node.id('dir')
7515 local GLYPH = node.id('glyph')
7516
7517 local function insert_implicit(head, state, outer)
7518   local new_state = state
7519   if state.sim and state.eim and state.sim ~= state.eim then
7520     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
7521     local d = node.new(DIR)
7522     d.dir = '+' .. dir
7523     node.insert_before(head, state.sim, d)
7524     local d = node.new(DIR)
7525     d.dir = '-' .. dir
7526     node.insert_after(head, state.eim, d)
7527   end
7528   new_state.sim, new_state.eim = nil, nil
7529   return head, new_state
7530 end
7531
7532 local function insert_numeric(head, state)
7533   local new
7534   local new_state = state
7535   if state.san and state.ean and state.san ~= state.ean then
7536     local d = node.new(DIR)
7537     d.dir = '+TLT'
7538     _, new = node.insert_before(head, state.san, d)
7539     if state.san == state.sim then state.sim = new end
7540     local d = node.new(DIR)
7541     d.dir = '-TLT'
7542     _, new = node.insert_after(head, state.ean, d)
7543     if state.ean == state.eim then state.eim = new end
7544   end
7545   new_state.san, new_state.ean = nil, nil
7546   return head, new_state
7547 end
7548
7549 -- TODO - \hbox with an explicit dir can lead to wrong results
7550 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7551 -- was s made to improve the situation, but the problem is the 3-dir
7552 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7553 -- well.
7554
7555 function Babel.bidi(head, ispar, hdir)
7556   local d -- d is used mainly for computations in a loop
7557   local prev_d = ''
```

```

7558 local new_d = false
7559
7560 local nodes = {}
7561 local outer_first = nil
7562 local inmath = false
7563
7564 local glue_d = nil
7565 local glue_i = nil
7566
7567 local has_en = false
7568 local first_et = nil
7569
7570 local has_hyperlink = false
7571
7572 local ATDIR = Babel.attr_dir
7573
7574 local save_outer
7575 local temp = node.get_attribute(head, ATDIR)
7576 if temp then
7577     temp = temp & 0x3
7578     save_outer = (temp == 0 and 'l') or
7579                 (temp == 1 and 'r') or
7580                 (temp == 2 and 'al')
7581 elseif ispar then -- Or error? Shouldn't happen
7582     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7583 else -- Or error? Shouldn't happen
7584     save_outer = ('TRT' == hdir) and 'r' or 'l'
7585 end
7586 -- when the callback is called, we are just _after_ the box,
7587 -- and the textdir is that of the surrounding text
7588 -- if not ispar and hdir ~= tex.textdir then
7589 --     save_outer = ('TRT' == hdir) and 'r' or 'l'
7590 -- end
7591 local outer = save_outer
7592 local last = outer
7593 -- 'al' is only taken into account in the first, current loop
7594 if save_outer == 'al' then save_outer = 'r' end
7595
7596 local fontmap = Babel.fontmap
7597
7598 for item in node.traverse(head) do
7599
7600     -- In what follows, #node is the last (previous) node, because the
7601     -- current one is not added until we start processing the neutrals.
7602
7603     -- three cases: glyph, dir, otherwise
7604     if item.id == GLYPH
7605         or (item.id == 7 and item.subtype == 2) then
7606
7607         local d_font = nil
7608         local item_r
7609         if item.id == 7 and item.subtype == 2 then
7610             item_r = item.replace -- automatic discs have just 1 glyph
7611         else
7612             item_r = item
7613         end
7614         local chardata = characters[item_r.char]
7615         d = chardata and chardata.d or nil
7616         if not d or d == 'nsm' then
7617             for nn, et in ipairs(ranges) do
7618                 if item_r.char < et[1] then
7619                     break
7620                 elseif item_r.char <= et[2] then

```



```

7621         if not d then d = et[3]
7622         elseif d == 'nsm' then d_font = et[3]
7623         end
7624         break
7625     end
7626 end
7627 end
7628 d = d or 'l'
7629
7630 -- A short 'pause' in bidi for mapfont
7631 d_font = d_font or d
7632 d_font = (d_font == 'l' and 0) or
7633         (d_font == 'nsm' and 0) or
7634         (d_font == 'r' and 1) or
7635         (d_font == 'al' and 2) or
7636         (d_font == 'an' and 2) or nil
7637 if d_font and fontmap and fontmap[d_font][item_r.font] then
7638     item_r.font = fontmap[d_font][item_r.font]
7639 end
7640
7641 if new_d then
7642     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7643     if inmath then
7644         attr_d = 0
7645     else
7646         attr_d = node.get_attribute(item, ATDIR)
7647         attr_d = attr_d & 0x3
7648     end
7649     if attr_d == 1 then
7650         outer_first = 'r'
7651         last = 'r'
7652     elseif attr_d == 2 then
7653         outer_first = 'r'
7654         last = 'al'
7655     else
7656         outer_first = 'l'
7657         last = 'l'
7658     end
7659     outer = last
7660     has_en = false
7661     first_et = nil
7662     new_d = false
7663 end
7664
7665 if glue_d then
7666     if (d == 'l' and 'l' or 'r') ~= glue_d then
7667         table.insert(nodes, {glue_i, 'on', nil})
7668     end
7669     glue_d = nil
7670     glue_i = nil
7671 end
7672
7673 elseif item.id == DIR then
7674     d = nil
7675
7676     if head ~= item then new_d = true end
7677
7678 elseif item.id == node.id'glue' and item.subtype == 13 then
7679     glue_d = d
7680     glue_i = item
7681     d = nil
7682
7683 elseif item.id == node.id'math' then

```

```

7684     inmath = (item.subtype == 0)
7685
7686     elseif item.id == 8 and item.subtype == 19 then
7687         has_hyperlink = true
7688
7689     else
7690         d = nil
7691     end
7692
7693     -- AL <= EN/ET/ES      -- W2 + W3 + W6
7694     if last == 'al' and d == 'en' then
7695         d = 'an'          -- W3
7696     elseif last == 'al' and (d == 'et' or d == 'es') then
7697         d = 'on'          -- W6
7698     end
7699
7700     -- EN + CS/ES + EN      -- W4
7701     if d == 'en' and #nodes >= 2 then
7702         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7703             and nodes[#nodes-1][2] == 'en' then
7704             nodes[#nodes][2] = 'en'
7705         end
7706     end
7707
7708     -- AN + CS + AN        -- W4 too, because uax9 mixes both cases
7709     if d == 'an' and #nodes >= 2 then
7710         if (nodes[#nodes][2] == 'cs')
7711             and nodes[#nodes-1][2] == 'an' then
7712             nodes[#nodes][2] = 'an'
7713         end
7714     end
7715
7716     -- ET/EN                -- W5 + W7->l / W6->on
7717     if d == 'et' then
7718         first_et = first_et or (#nodes + 1)
7719     elseif d == 'en' then
7720         has_en = true
7721         first_et = first_et or (#nodes + 1)
7722     elseif first_et then      -- d may be nil here !
7723         if has_en then
7724             if last == 'l' then
7725                 temp = 'l'    -- W7
7726             else
7727                 temp = 'en'  -- W5
7728             end
7729         else
7730             temp = 'on'      -- W6
7731         end
7732         for e = first_et, #nodes do
7733             if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7734         end
7735         first_et = nil
7736         has_en = false
7737     end
7738
7739     -- Force mathdir in math if ON (currently works as expected only
7740     -- with 'l')
7741     if inmath and d == 'on' then
7742         d = ('TRT' == tex.mathdir) and 'r' or 'l'
7743     end
7744
7745     if d then
7746         if d == 'al' then

```

```

7747     d = 'r'
7748     last = 'al'
7749     elseif d == 'l' or d == 'r' then
7750         last = d
7751     end
7752     prev_d = d
7753     table.insert(nodes, {item, d, outer_first})
7754 end
7755
7756 outer_first = nil
7757
7758 end
7759
7760 -- TODO -- repeated here in case EN/ET is the last node. Find a
7761 -- better way of doing things:
7762 if first_et then      -- dir may be nil here !
7763     if has_en then
7764         if last == 'l' then
7765             temp = 'l'    -- W7
7766         else
7767             temp = 'en'  -- W5
7768         end
7769     else
7770         temp = 'on'     -- W6
7771     end
7772     for e = first_et, #nodes do
7773         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7774     end
7775 end
7776
7777 -- dummy node, to close things
7778 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7779
7780 ----- NEUTRAL -----
7781
7782 outer = save_outer
7783 last = outer
7784
7785 local first_on = nil
7786
7787 for q = 1, #nodes do
7788     local item
7789
7790     local outer_first = nodes[q][3]
7791     outer = outer_first or outer
7792     last = outer_first or last
7793
7794     local d = nodes[q][2]
7795     if d == 'an' or d == 'en' then d = 'r' end
7796     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7797
7798     if d == 'on' then
7799         first_on = first_on or q
7800     elseif first_on then
7801         if last == d then
7802             temp = d
7803         else
7804             temp = outer
7805         end
7806         for r = first_on, q - 1 do
7807             nodes[r][2] = temp
7808             item = nodes[r][1]    -- MIRRORING
7809             if Babel.mirroring_enabled and item.id == GLYPH

```

```

7810         and temp == 'r' and characters[item.char] then
7811         local font_mode = ''
7812         if item.font > 0 and font.fonts[item.font].properties then
7813             font_mode = font.fonts[item.font].properties.mode
7814         end
7815         if font_mode ~= 'harf' and font_mode ~= 'plug' then
7816             item.char = characters[item.char].m or item.char
7817         end
7818     end
7819 end
7820 first_on = nil
7821 end
7822
7823 if d == 'r' or d == 'l' then last = d end
7824 end
7825
7826 ----- IMPLICIT, REORDER -----
7827
7828 outer = save_outer
7829 last = outer
7830
7831 local state = {}
7832 state.has_r = false
7833
7834 for q = 1, #nodes do
7835
7836     local item = nodes[q][1]
7837
7838     outer = nodes[q][3] or outer
7839
7840     local d = nodes[q][2]
7841
7842     if d == 'nsm' then d = last end          -- W1
7843     if d == 'en' then d = 'an' end
7844     local isdir = (d == 'r' or d == 'l')
7845
7846     if outer == 'l' and d == 'an' then
7847         state.san = state.san or item
7848         state.ean = item
7849     elseif state.san then
7850         head, state = insert_numeric(head, state)
7851     end
7852
7853     if outer == 'l' then
7854         if d == 'an' or d == 'r' then      -- im -> implicit
7855             if d == 'r' then state.has_r = true end
7856             state.sim = state.sim or item
7857             state.eim = item
7858         elseif d == 'l' and state.sim and state.has_r then
7859             head, state = insert_implicit(head, state, outer)
7860         elseif d == 'l' then
7861             state.sim, state.eim, state.has_r = nil, nil, false
7862         end
7863     else
7864         if d == 'an' or d == 'l' then
7865             if nodes[q][3] then -- nil except after an explicit dir
7866                 state.sim = item -- so we move sim 'inside' the group
7867             else
7868                 state.sim = state.sim or item
7869             end
7870             state.eim = item
7871         elseif d == 'r' and state.sim then
7872             head, state = insert_implicit(head, state, outer)

```

```

7873     elseif d == 'r' then
7874         state.sim, state.eim = nil, nil
7875     end
7876 end
7877
7878 if isdir then
7879     last = d           -- Don't search back - best save now
7880 elseif d == 'on' and state.san then
7881     state.san = state.san or item
7882     state.ean = item
7883 end
7884
7885 end
7886
7887 head = node.prev(head) or head
7888
7889 ----- FIX HYPERLINKS -----
7890
7891 if has_hyperlink then
7892     local flag, linking = 0, 0
7893     for item in node.traverse(head) do
7894         if item.id == DIR then
7895             if item.dir == '+TRT' or item.dir == '+TLT' then
7896                 flag = flag + 1
7897             elseif item.dir == '-TRT' or item.dir == '-TLT' then
7898                 flag = flag - 1
7899             end
7900             elseif item.id == 8 and item.subtype == 19 then
7901                 linking = flag
7902             elseif item.id == 8 and item.subtype == 20 then
7903                 if linking > 0 then
7904                     if item.prev.id == DIR and
7905                         (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
7906                         d = node.new(DIR)
7907                         d.dir = item.prev.dir
7908                         node.remove(head, item.prev)
7909                         node.insert_after(head, item, d)
7910                     end
7911                 end
7912                 linking = 0
7913             end
7914         end
7915     end
7916
7917 return head
7918 end
7919 </basic>

```

11 Data for CJK

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

```

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

```

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

12 The ‘nil’ language

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

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

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

```
7920 ⟨*nil⟩
7921 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ v⟨⟨version⟩⟩ Nil language]
7922 \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.

```
7923 \ifx\l@nil\@undefined
7924   \newlanguage\l@nil
7925   \@namedef{bbl@hyphendata@the\l@nil}{\{}}% Remove warning
7926   \let\bbl@elt\relax
7927   \edef\bbl@languages{% Add it to the list of languages
7928     \bbl@languages\bbl@elt{nil}{\the\l@nil}{\{}}
7929 \fi
```

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

```
7930 \providehyphenmins{\CurrentOption}{\m@ne@m@ne}
```

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

```
\captionnil
\datenil
7931 \let\captionnil\@empty
7932 \let\datenil\@empty
```

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

```
7933 \def\bbl@inidata@nil{%
7934   \bbl@elt{identification}{tag.ini}{und}%
7935   \bbl@elt{identification}{load.level}{0}%
7936   \bbl@elt{identification}{charset}{utf8}%
7937   \bbl@elt{identification}{version}{1.0}%
7938   \bbl@elt{identification}{date}{2022-05-16}%
7939   \bbl@elt{identification}{name.local}{nil}%
7940   \bbl@elt{identification}{name.english}{nil}%
7941   \bbl@elt{identification}{name.babel}{nil}%
7942   \bbl@elt{identification}{tag.bcp47}{und}%
7943   \bbl@elt{identification}{language.tag.bcp47}{und}%
7944   \bbl@elt{identification}{tag.opentype}{dflt}%
7945   \bbl@elt{identification}{script.name}{Latin}%
7946   \bbl@elt{identification}{script.tag.bcp47}{Latn}%
7947   \bbl@elt{identification}{script.tag.opentype}{DFLT}%
7948   \bbl@elt{identification}{level}{1}%
7949   \bbl@elt{identification}{encodings}{}%
7950   \bbl@elt{identification}{derivate}{no}}
7951 \@namedef{bbl@tbc@nil}{und}
7952 \@namedef{bbl@lbc@nil}{und}
7953 \@namedef{bbl@casing@nil}{und} % TODO
7954 \@namedef{bbl@lotf@nil}{dflt}
7955 \@namedef{bbl@elname@nil}{nil}
7956 \@namedef{bbl@lname@nil}{nil}
7957 \@namedef{bbl@esname@nil}{Latin}
7958 \@namedef{bbl@sname@nil}{Latin}
7959 \@namedef{bbl@sbc@nil}{Latn}
7960 \@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.

```
7961 \ldf@finish{nil}
7962 ⟨/nil⟩
```

13 Calendars

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

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

```
7963 <<*Compute Julian day>> ≡
7964 \def\bbl@fpmo#1#2{(#1-#2*floor(#1/#2))}
7965 \def\bbl@cs@gregleap#1{%
7966   (\bbl@fpmo{#1}{4} == 0) &&
7967   (!( \bbl@fpmo{#1}{100} == 0) && (\bbl@fpmo{#1}{400} != 0))}
7968 \def\bbl@cs@jd#1#2#3{% year, month, day
7969   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
7970     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
7971     floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) +
7972     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3} }
7973 <</Compute Julian day>>
```

13.1 Islamic

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

```
7974 (*ca-islamic)
7975 \ExplSyntaxOn
7976 <<Compute Julian day>>
7977 % == islamic (default)
7978 % Not yet implemented
7979 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{}
```

The Civil calendar.

```
7980 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
7981   ((#3 + ceil(29.5 * (#2 - 1)) +
7982     (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
7983     1948439.5) - 1) }
7984 \@namedef{\bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
7985 \@namedef{\bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
7986 \@namedef{\bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
7987 \@namedef{\bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
7988 \@namedef{\bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
7989 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
7990   \edef\bbl@tempa{%
7991     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
7992   \edef#5{%
7993     \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
7994   \edef#6{\fp_eval:n{
7995     min(12, ceil((\bbl@tempa - (29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
7996   \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).

```
7997 \def\bbl@cs@umalqura@data{56660, 56690, 56719, 56749, 56778, 56808, %
7998 56837, 56867, 56897, 56926, 56956, 56985, 57015, 57044, 57074, 57103, %
7999 57133, 57162, 57192, 57221, 57251, 57280, 57310, 57340, 57369, 57399, %
8000 57429, 57458, 57487, 57517, 57546, 57576, 57605, 57634, 57664, 57694, %
8001 57723, 57753, 57783, 57813, 57842, 57871, 57901, 57930, 57959, 57989, %
8002 58018, 58048, 58077, 58107, 58137, 58167, 58196, 58226, 58255, 58285, %
8003 58314, 58343, 58373, 58402, 58432, 58461, 58491, 58521, 58551, 58580, %
8004 58610, 58639, 58669, 58698, 58727, 58757, 58786, 58816, 58845, 58875, %
8005 58905, 58934, 58964, 58994, 59023, 59053, 59082, 59111, 59141, 59170, %
8006 59200, 59229, 59259, 59288, 59318, 59348, 59377, 59407, 59436, 59466, %
8007 59495, 59525, 59554, 59584, 59613, 59643, 59672, 59702, 59731, 59761, %
8008 59791, 59820, 59850, 59879, 59909, 59939, 59968, 59997, 60027, 60056, %
```

```

8009 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
8010 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8011 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8012 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8013 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
8014 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
8015 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8016 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8017 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8018 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8019 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8020 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
8021 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8022 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8023 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
8024 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8025 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8026 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8027 65401,65431,65460,65490,65520}
8028 \namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
8029 \namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
8030 \namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
8031 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
8032 \ifnum#2>2014 \ifnum#2<2038
8033 \bbl@afterfi\expandafter\@gobble
8034 \fi\fi
8035 {\bbl@error{Year~out~of~range}{The~allowed~range~is~2014-2038}}%
8036 \def\bbl@tempd{\fp_eval:n{ % (Julian) day
8037 \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8038 \count@\@ne
8039 \bbl@foreach\bbl@cs@umalqura@data{%
8040 \advance\count@\@ne
8041 \ifnum##1>\bbl@tempd\else
8042 \edef\bbl@tempe{\the\count@}%
8043 \edef\bbl@tempb{##1}%
8044 \fi}%
8045 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
8046 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1) / 12) }}% annus
8047 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
8048 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
8049 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}}
8050 \ExplSyntaxOff
8051 \bbl@add\bbl@precalendar{%
8052 \bbl@replace\bbl@ld@calendar{-civil}{}%
8053 \bbl@replace\bbl@ld@calendar{-umalqura}{}%
8054 \bbl@replace\bbl@ld@calendar{+}{}%
8055 \bbl@replace\bbl@ld@calendar{-}{}}
8056 \ca-islamic)

```

13.2 Hebrew

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

```

8057 (*ca-hebrew)
8058 \newcount\bbl@cntcommon
8059 \def\bbl@remainder#1#2#3{%
8060 #3=#1\relax
8061 \divide #3 by #2\relax
8062 \multiply #3 by -#2\relax
8063 \advance #3 by #1\relax}%
8064 \newif\ifbbl@divisible
8065 \def\bbl@checkifdivisible#1#2{%

```



```

8066 {\countdef\tmp=0
8067 \bbl@remainder{#1}{#2}{\tmp}%
8068 \ifnum \tmp=0
8069 \global\bbl@divisibletrue
8070 \else
8071 \global\bbl@divisiblefalse
8072 \fi}}
8073 \newif\ifbbl@gregleap
8074 \def\bbl@ifgregleap#1{%
8075 \bbl@checkifdivisible{#1}{4}%
8076 \ifbbl@divisible
8077 \bbl@checkifdivisible{#1}{100}%
8078 \ifbbl@divisible
8079 \bbl@checkifdivisible{#1}{400}%
8080 \ifbbl@divisible
8081 \bbl@gregleaptrue
8082 \else
8083 \bbl@gregleapfalse
8084 \fi
8085 \else
8086 \bbl@gregleaptrue
8087 \fi
8088 \else
8089 \bbl@gregleapfalse
8090 \fi
8091 \ifbbl@gregleap}
8092 \def\bbl@gregdayspriormonths#1#2#3{%
8093 {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
8094 181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
8095 \bbl@ifgregleap{#2}%
8096 \ifnum #1 > 2
8097 \advance #3 by 1
8098 \fi
8099 \fi
8100 \global\bbl@cntcommon=#3}%
8101 #3=\bbl@cntcommon}
8102 \def\bbl@gregdaysprioryears#1#2{%
8103 {\countdef\tmpc=4
8104 \countdef\tmpb=2
8105 \tmpb=#1\relax
8106 \advance \tmpb by -1
8107 \tmpc=\tmpb
8108 \multiply \tmpc by 365
8109 #2=\tmpc
8110 \tmpc=\tmpb
8111 \divide \tmpc by 4
8112 \advance #2 by \tmpc
8113 \tmpc=\tmpb
8114 \divide \tmpc by 100
8115 \advance #2 by -\tmpc
8116 \tmpc=\tmpb
8117 \divide \tmpc by 400
8118 \advance #2 by \tmpc
8119 \global\bbl@cntcommon=#2\relax}%
8120 #2=\bbl@cntcommon}
8121 \def\bbl@absfromgreg#1#2#3#4{%
8122 {\countdef\tmpd=0
8123 #4=#1\relax
8124 \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
8125 \advance #4 by \tmpd
8126 \bbl@gregdaysprioryears{#3}{\tmpd}%
8127 \advance #4 by \tmpd
8128 \global\bbl@cntcommon=#4\relax}%

```

```

8129 #4=\bbl@cntcommon}
8130 \newif\ifbbl@hebrleap
8131 \def\bbl@checkleaphebryear#1{%
8132   {\countdef\tmpa=0
8133    \countdef\tmpb=1
8134    \tmpa=#1\relax
8135    \multiply\tmpa by 7
8136    \advance\tmpa by 1
8137    \bbl@remainder{\tmpa}{19}{\tmpb}%
8138    \ifnum\tmpb < 7
8139      \global\bbl@hebrleaptrue
8140    \else
8141      \global\bbl@hebrleapfalse
8142    \fi}}
8143 \def\bbl@hebrlapsedmonths#1#2{%
8144   {\countdef\tmpa=0
8145    \countdef\tmpb=1
8146    \countdef\tmpc=2
8147    \tmpa=#1\relax
8148    \advance\tmpa by -1
8149    #2=\tmpa
8150    \divide#2 by 19
8151    \multiply#2 by 235
8152    \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8153    \tmpc=\tmpb
8154    \multiply\tmpb by 12
8155    \advance#2 by \tmpb
8156    \multiply\tmpc by 7
8157    \advance\tmpc by 1
8158    \divide\tmpc by 19
8159    \advance#2 by \tmpc
8160    \global\bbl@cntcommon=#2}%
8161   #2=\bbl@cntcommon}
8162 \def\bbl@hebrlapseddays#1#2{%
8163   {\countdef\tmpa=0
8164    \countdef\tmpb=1
8165    \countdef\tmpc=2
8166    \bbl@hebrlapsedmonths{#1}{#2}%
8167    \tmpa=#2\relax
8168    \multiply\tmpa by 13753
8169    \advance\tmpa by 5604
8170    \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8171    \divide\tmpa by 25920
8172    \multiply#2 by 29
8173    \advance#2 by 1
8174    \advance#2 by \tmpa
8175    \bbl@remainder{#2}{7}{\tmpa}%
8176    \ifnum\tmpc < 19440
8177      \ifnum\tmpc < 9924
8178        \else
8179          \ifnum\tmpa=2
8180            \bbl@checkleaphebryear{#1}% of a common year
8181            \ifbbl@hebrleap
8182              \else
8183                \advance#2 by 1
8184              \fi
8185            \fi
8186          \fi
8187          \ifnum\tmpc < 16789
8188            \else
8189              \ifnum\tmpa=1
8190                \advance#1 by -1
8191                \bbl@checkleaphebryear{#1}% at the end of leap year

```

```

8192         \ifbbl@hebrleap
8193         \advance #2 by 1
8194     \fi
8195 \fi
8196 \fi
8197 \else
8198     \advance #2 by 1
8199 \fi
8200 \bbl@remainder{#2}{7}{\tmpa}%
8201 \ifnum \tmpa=0
8202     \advance #2 by 1
8203 \else
8204     \ifnum \tmpa=3
8205         \advance #2 by 1
8206     \else
8207         \ifnum \tmpa=5
8208             \advance #2 by 1
8209         \fi
8210     \fi
8211 \fi
8212 \global\bbl@cntcommon=#2\relax}%
8213 #2=\bbl@cntcommon}
8214 \def\bbl@daysinhebryear#1#2{%
8215     {\countdef\tmpe=12
8216     \bbl@hebrlapseddays{#1}{\tmpe}%
8217     \advance #1 by 1
8218     \bbl@hebrlapseddays{#1}{#2}%
8219     \advance #2 by -\tmpe
8220     \global\bbl@cntcommon=#2}%
8221 #2=\bbl@cntcommon}
8222 \def\bbl@hebrdayspriormonths#1#2#3{%
8223     {\countdef\tmpf= 14
8224     #3=\ifcase #1\relax
8225         0 \or
8226         0 \or
8227         30 \or
8228         59 \or
8229         89 \or
8230         118 \or
8231         148 \or
8232         148 \or
8233         177 \or
8234         207 \or
8235         236 \or
8236         266 \or
8237         295 \or
8238         325 \or
8239         400
8240     \fi
8241     \bbl@checkleaphebryear{#2}%
8242     \ifbbl@hebrleap
8243         \ifnum #1 > 6
8244             \advance #3 by 30
8245         \fi
8246     \fi
8247     \bbl@daysinhebryear{#2}{\tmpf}%
8248     \ifnum #1 > 3
8249         \ifnum \tmpf=353
8250             \advance #3 by -1
8251         \fi
8252         \ifnum \tmpf=383
8253             \advance #3 by -1
8254         \fi

```

```

8255 \fi
8256 \ifnum #1 > 2
8257     \ifnum \tmpf=355
8258         \advance #3 by 1
8259     \fi
8260     \ifnum \tmpf=385
8261         \advance #3 by 1
8262     \fi
8263 \fi
8264 \global\bbl@cntcommon=#3\relax}%
8265 #3=\bbl@cntcommon}
8266 \def\bbl@absfromhebr#1#2#3#4{%
8267     {#4=#1\relax
8268     \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8269     \advance #4 by #1\relax
8270     \bbl@hebrrelapseddays{#3}{#1}%
8271     \advance #4 by #1\relax
8272     \advance #4 by -1373429
8273     \global\bbl@cntcommon=#4\relax}%
8274 #4=\bbl@cntcommon}
8275 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8276     {\countdef\tmpx= 17
8277     \countdef\tmpy= 18
8278     \countdef\tmpz= 19
8279     #6=#3\relax
8280     \global\advance #6 by 3761
8281     \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8282     \tmpz=1 \tmpy=1
8283     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8284     \ifnum \tmpx > #4\relax
8285         \global\advance #6 by -1
8286         \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8287     \fi
8288     \advance #4 by -\tmpx
8289     \advance #4 by 1
8290     #5=#4\relax
8291     \divide #5 by 30
8292     \loop
8293         \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8294         \ifnum \tmpx < #4\relax
8295             \advance #5 by 1
8296             \tmpy=\tmpx
8297         \repeat
8298     \global\advance #5 by -1
8299     \global\advance #4 by -\tmpy}}
8300 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
8301 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8302 \def\bbl@ca@hebrew#1-#2-#3\@#4#5#6{%
8303     \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8304     \bbl@hebrfromgreg
8305     {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8306     {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
8307     \edef#4{\the\bbl@hebryear}%
8308     \edef#5{\the\bbl@hebrmonth}%
8309     \edef#6{\the\bbl@hebrday}}
8310 </ca-hebrew>

```

13.3 Persian

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

pre-calculated and stored. Actually, all we need is the first day (either March 20 or March 21).

```

8311 (*ca-persian)
8312 \ExplSyntaxOn
8313 <<Compute Julian day>>
8314 \def\bbbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8315 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8316 \def\bbbl@ca@persian#1-#2-#3\@#4#5#6{%
8317 \edef\bbbl@tempa{#1}% 20XX-03-\bbbl@tempe = 1 farvardin:
8318 \ifnum\bbbl@tempa>2012 \ifnum\bbbl@tempa<2051
8319 \bbbl@afterfi\expandafter\@gobble
8320 \fi\fi
8321 {\bbbl@error{Year~out~of~range}{The~allowed~range~is~2013-2050}}%
8322 \bbbl@xin@{\bbbl@tempa}{\bbbl@cs@firstjal@xx}%
8323 \ifin@def\bbbl@tempe{20}\else\def\bbbl@tempe{21}\fi
8324 \edef\bbbl@tempc{\fp_eval:n{\bbbl@cs@jd{\bbbl@tempa}{#2}{#3}+.5}}% current
8325 \edef\bbbl@tempb{\fp_eval:n{\bbbl@cs@jd{\bbbl@tempa}{03}{\bbbl@tempe}+.5}}% begin
8326 \ifnum\bbbl@tempc<\bbbl@tempb
8327 \edef\bbbl@tempa{\fp_eval:n{\bbbl@tempa-1}}% go back 1 year and redo
8328 \bbbl@xin@{\bbbl@tempa}{\bbbl@cs@firstjal@xx}%
8329 \ifin@def\bbbl@tempe{20}\else\def\bbbl@tempe{21}\fi
8330 \edef\bbbl@tempb{\fp_eval:n{\bbbl@cs@jd{\bbbl@tempa}{03}{\bbbl@tempe}+.5}}%
8331 \fi
8332 \edef#4{\fp_eval:n{\bbbl@tempa-621}}% set Jalali year
8333 \edef#6{\fp_eval:n{\bbbl@tempc-\bbbl@tempb+1}}% days from 1 farvardin
8334 \edef#5{\fp_eval:n{% set Jalali month
8335 (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8336 \edef#6{\fp_eval:n{% set Jalali day
8337 (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6))}}
8338 \ExplSyntaxOff
8339 </ca-persian>

```

13.4 Coptic and Ethiopic

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

```

8340 (*ca-coptic)
8341 \ExplSyntaxOn
8342 <<Compute Julian day>>
8343 \def\bbbl@ca@coptic#1-#2-#3\@#4#5#6{%
8344 \edef\bbbl@tempd{\fp_eval:n{floor(\bbbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8345 \edef\bbbl@tempc{\fp_eval:n{\bbbl@tempd - 1825029.5}}%
8346 \edef#4{\fp_eval:n{%
8347 floor((\bbbl@tempc - floor((\bbbl@tempc+366) / 1461)) / 365) + 1}}%
8348 \edef\bbbl@tempc{\fp_eval:n{%
8349 \bbbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8350 \edef#5{\fp_eval:n{floor(\bbbl@tempc / 30) + 1}}%
8351 \edef#6{\fp_eval:n{\bbbl@tempc - (#5 - 1) * 30 + 1}}
8352 \ExplSyntaxOff
8353 </ca-coptic>
8354 (*ca-ethiopic)
8355 \ExplSyntaxOn
8356 <<Compute Julian day>>
8357 \def\bbbl@ca@ethiopic#1-#2-#3\@#4#5#6{%
8358 \edef\bbbl@tempd{\fp_eval:n{floor(\bbbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8359 \edef\bbbl@tempc{\fp_eval:n{\bbbl@tempd - 1724220.5}}%
8360 \edef#4{\fp_eval:n{%
8361 floor((\bbbl@tempc - floor((\bbbl@tempc+366) / 1461)) / 365) + 1}}%
8362 \edef\bbbl@tempc{\fp_eval:n{%
8363 \bbbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8364 \edef#5{\fp_eval:n{floor(\bbbl@tempc / 30) + 1}}%
8365 \edef#6{\fp_eval:n{\bbbl@tempc - (#5 - 1) * 30 + 1}}
8366 \ExplSyntaxOff

```

8367 </ca-ethiopic>

13.5 Buddhist

That's very simple.

```
8368 (*ca-buddhist)
8369 \def\bbbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
8370   \edef#4{\number\numexpr#1+543\relax}%
8371   \edef#5{#2}%
8372   \edef#6{#3}}
8373 </ca-buddhist)
8374 %
8375 % \subsection{Chinese}
8376 %
8377 % Brute force, with the Julian day of first day of each month. The
8378 % table has been computed with the help of \textsf{python-lunardate} by
8379 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
8380 % is 2015-2044.
8381 %
8382 % \begin{macrocode}
8383 (*ca-chinese)
8384 \ExplSyntaxOn
8385 <<Compute Julian day>>
8386 \def\bbbl@ca@chinese#1-#2-#3\@@#4#5#6{%
8387   \edef\bbbl@tempd{\fp_eval:n{%
8388     \bbbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
8389   \count@ \z@
8390   \@tempcnta=2015
8391   \bbbl@foreach\bbbl@cs@chinese@data{%
8392     \ifnum##1>\bbbl@tempd\else
8393       \advance\count@\@ne
8394       \ifnum\count@>12
8395         \count@\@ne
8396         \advance\@tempcnta\@ne\fi
8397       \bbbl@xin@{,##1,}{,\bbbl@cs@chinese@leap,}%
8398       \ifin@
8399         \advance\count@\m@ne
8400       \edef\bbbl@tempe{\the\numexpr\count@+12\relax}%
8401       \else
8402         \edef\bbbl@tempe{\the\count@}%
8403       \fi
8404       \edef\bbbl@tempb{##1}%
8405       \fi}%
8406   \edef#4{\the\@tempcnta}%
8407   \edef#5{\bbbl@tempe}%
8408   \edef#6{\the\numexpr\bbbl@tempd-\bbbl@tempb+1\relax}}
8409 \def\bbbl@cs@chinese@leap{%
8410   885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
8411 \def\bbbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
8412   354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
8413   768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
8414   1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
8415   1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
8416   1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
8417   2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
8418   2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
8419   2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
8420   3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
8421   3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
8422   3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
8423   4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
8424   4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
8425   5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
```

```

8426 5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
8427 5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
8428 6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
8429 6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
8430 6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
8431 7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
8432 7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
8433 7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
8434 8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8435 8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8436 8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
8437 9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
8438 9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
8439 10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
8440 10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
8441 10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
8442 10896,10926,10956,10986,11015,11045,11074,11103}
8443 \ExplSyntaxOff
8444 </ca-chinese>

```

14 Support for Plain T_EX (plain.def)

14.1 Not renaming hyphen.tex

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

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

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

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

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

```

8445 (*bplain | blplain)
8446 \catcode\{=1 % left brace is begin-group character
8447 \catcode\}=2 % right brace is end-group character
8448 \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).

```

8449 \openin 0 hyphen.cfg
8450 \ifeof0
8451 \else
8452 \let\input

```

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

```

8453 \def\input #1 {%
8454 \let\input\a
8455 \a hyphen.cfg
8456 \let\a\undefined
8457 }
8458 \fi
8459 </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`.

```
8460 (bplain)\a plain.tex
8461 (bplain)\a lplain.tex
```

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

```
8462 (bplain)\def\fmtname{babel-plain}
8463 (bplain)\def\fmtname{babel-lplain}
```

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

14.2 Emulating some \LaTeX features

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

```
8464 (<*Emulate LaTeX) ≡
8465 \def\@empty{}
8466 \def\loadlocalcfg#1{%
8467   \openin0#1.cfg
8468   \ifeof0
8469     \closein0
8470   \else
8471     \closein0
8472     {\immediate\writel6{*****}%
8473      \immediate\writel6{* Local config file #1.cfg used}%
8474      \immediate\writel6{*}%
8475      }
8476     \input #1.cfg\relax
8477   \fi
8478   \@endofldf}
```

14.3 General tools

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

```
8479 \long\def\@firstofone#1{#1}
8480 \long\def\@firstoftwo#1#2{#1}
8481 \long\def\@secondoftwo#1#2{#2}
8482 \def\@nnil{\@nil}
8483 \def\@gobbletwo#1#2{}
8484 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
8485 \def\@star@or@long#1{%
8486   \@ifstar
8487   {\let\@ngrel@x\relax#1}%
8488   {\let\@ngrel@x\long#1}}
8489 \let\@ngrel@x\relax
8490 \def\@car#1#2\@nil{#1}
8491 \def\@cdr#1#2\@nil{#2}
8492 \let\@typeset@protect\relax
8493 \let\protected@edef\edef
8494 \long\def\@gobble#1{}
8495 \edef\@backslashchar{\expandafter\@gobble\string\}
8496 \def\strip@prefix#1>{}
8497 \def@gaddto@macro#1#2{{%
8498   \toks@\expandafter{#1#2}%
8499   \xdef#1{\the\toks@}}}
8500 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8501 \def\@nameuse#1{\csname #1\endcsname}
```



```

8502 \def\@ifundefined#1{%
8503   \expandafter\ifx\csname#1\endcsname\relax
8504     \expandafter\@firstoftwo
8505   \else
8506     \expandafter\@secondoftwo
8507   \fi}
8508 \def\@expandtwoargs#1#2#3{%
8509   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8510 \def\zap@space#1 #2{%
8511   #1%
8512   \ifx#2\@empty\else\expandafter\zap@space\fi
8513   #2}
8514 \let\bbl@trace\@gobble
8515 \def\bbl@error#1#2{%
8516   \begingroup
8517     \newlinechar=`^^J
8518     \def\{^^J(babel) }%
8519     \errhelp{#2}\errmessage{\#1}%
8520   \endgroup}
8521 \def\bbl@warning#1{%
8522   \begingroup
8523     \newlinechar=`^^J
8524     \def\{^^J(babel) }%
8525     \message{\#1}%
8526   \endgroup}
8527 \let\bbl@infowarn\bbl@warning
8528 \def\bbl@info#1{%
8529   \begingroup
8530     \newlinechar=`^^J
8531     \def\{^^J}%
8532     \wlog{#1}%
8533   \endgroup}

```

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

```

8534 \ifx\@preamblecmds\undefined
8535   \def\@preamblecmds{}
8536 \fi
8537 \def\@onlypreamble#1{%
8538   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8539     \@preamblecmds\do#1}}
8540 \@onlypreamble\@onlypreamble

```

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

```

8541 \def\begindocument{%
8542   \@begindocumenthook
8543   \global\let\@begindocumenthook\undefined
8544   \def\do##1{\global\let##1\undefined}%
8545   \@preamblecmds
8546   \global\let\do\noexpand}
8547 \ifx\@begindocumenthook\undefined
8548   \def\@begindocumenthook{}
8549 \fi
8550 \@onlypreamble\@begindocumenthook
8551 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

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

```

8552 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
8553 \@onlypreamble\AtEndOfPackage
8554 \def\@endofldf{}
8555 \@onlypreamble\@endofldf
8556 \let\bbl@afterlang\@empty
8557 \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.

```
8558 \catcode`\&=\z@
8559 \ifx&if@filesw\undefined
8560 \expandafter\let\csname if@filesw\expandafter\endcsname
8561 \csname iffalse\endcsname
8562 \fi
8563 \catcode`\&=4
```

Mimic \LaTeX 's commands to define control sequences.

```
8564 \def\newcommand{\@star@or@long\new@command}
8565 \def\new@command#1{%
8566 \@testopt{\@newcommand#1}0}
8567 \def\@newcommand#1[#2]{%
8568 \ifnextchar [{\@xargdef#1[#2]}%
8569 {\@argdef#1[#2]}}
8570 \long\def\@argdef#1[#2]#3{%
8571 \@yargdef#1\@ne{#2}{#3}}
8572 \long\def\@xargdef#1[#2][#3]#4{%
8573 \expandafter\def\expandafter#1\expandafter{%
8574 \expandafter\@protected@testopt\expandafter #1%
8575 \csname\string#1\expandafter\endcsname{#3}}%
8576 \expandafter\@yargdef \csname\string#1\endcsname
8577 \tw@{#2}{#4}}
8578 \long\def\@yargdef#1#2#3{%
8579 \@tempcnta#3\relax
8580 \advance \@tempcnta \@ne
8581 \let\@hash@\relax
8582 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
8583 \@tempcntb #2%
8584 \@whilenum\@tempcntb <\@tempcnta
8585 \do{%
8586 \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
8587 \advance\@tempcntb \@ne}%
8588 \let\@hash@##%
8589 \l@ngrel\x\expandafter\def\expandafter#1\reserved@a}
8590 \def\providecommand{\@star@or@long\provide@command}
8591 \def\provide@command#1{%
8592 \begingroup
8593 \escapechar\m@ne\xdef\@gtempa{\string#1}%
8594 \endgroup
8595 \expandafter\@ifundefined\@gtempa
8596 {\def\reserved@a{\new@command#1}}%
8597 {\let\reserved@a\relax
8598 \def\reserved@a{\new@command\reserved@a}}%
8599 \reserved@a}%

8600 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
8601 \def\declare@robustcommand#1{%
8602 \edef\reserved@a{\string#1}%
8603 \def\reserved@b{#1}%
8604 \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
8605 \edef#1{%
8606 \ifx\reserved@a\reserved@b
8607 \noexpand\x@protect
8608 \noexpand#1%
8609 \fi
8610 \noexpand\protect
8611 \expandafter\noexpand\csname
8612 \expandafter\@gobble\string#1 \endcsname
8613 }%
8614 \expandafter\new@command\csname
8615 \expandafter\@gobble\string#1 \endcsname
```

```

8616 }
8617 \def\x@protect#1{%
8618   \ifx\protect\@typeset@protect\else
8619     \@x@protect#1%
8620   \fi
8621 }
8622 \catcode`\&=\z@ % Trick to hide conditionals
8623 \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`.

```

8624 \def\bbl@tempa{\csname newif\endcsname&ifin@}
8625 \catcode`\&=4
8626 \ifx\in@\undefined
8627   \def\in@#1#2{%
8628     \def\in@@##1#1##2##3\in@@{%
8629       \ifx\in@@##2\in@false\else\in@true\fi}%
8630     \in@@#2#1\in@\in@@}
8631 \else
8632   \let\bbl@tempa\@empty
8633 \fi
8634 \bbl@tempa

```

\LaTeX has a macro to check whether a certain package was loaded with specific options. The command has two extra arguments which are code to be executed in either the true or false case. This is used to detect whether the document needs one of the accents to be activated (`activegrave` and `activeacute`). For plain \TeX we assume that the user wants them to be active by default. Therefore the only thing we do is execute the third argument (the code for the true case).

```
8635 \def\@ifpackagewith#1#2#3#4{#3}
```

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

```
8636 \def\@ifl@aded#1#2#3#4{}
```

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

```

8637 \ifx\@tempcnta\undefined
8638   \csname newcount\endcsname\@tempcnta\relax
8639 \fi
8640 \ifx\@tempcntb\undefined
8641   \csname newcount\endcsname\@tempcntb\relax
8642 \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`).

```

8643 \ifx\bye\undefined
8644   \advance\count10 by -2\relax
8645 \fi
8646 \ifx\@ifnextchar\undefined
8647   \def\@ifnextchar#1#2#3{%
8648     \let\reserved@d=#1%
8649     \def\reserved@a{#2}\def\reserved@b{#3}%
8650     \futurelet\@let@token\@ifnch}
8651   \def\@ifnch{%
8652     \ifx\@let@token\@sptoken
8653       \let\reserved@c\@xifnch
8654     \else
8655       \ifx\@let@token\reserved@d
8656         \let\reserved@c\reserved@a
8657       \else
8658         \let\reserved@c\reserved@b
8659     \fi

```

```

8660 \fi
8661 \reserved@c}
8662 \def\:\let\sptoken= } \: % this makes \sptoken a space token
8663 \def\:\xifnch} \expandafter\def\:\{ \futurelet\@let@token\@ifnch}
8664 \fi
8665 \def\@testopt#1#2{%
8666 \ifnextchar[#{#1}{#1[#2]}}
8667 \def\@protected@testopt#1{%
8668 \ifx\protect\@typeset@protect
8669 \expandafter\@testopt
8670 \else
8671 \@x@protect#1%
8672 \fi}
8673 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8674 #2\relax}\fi}
8675 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8676 \else\expandafter\@gobble\fi{#1}}

```

14.4 Encoding related macros

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

```

8677 \def\DeclareTextCommand{%
8678 \@dec@text@cmd\providecommand
8679 }
8680 \def\ProvideTextCommand{%
8681 \@dec@text@cmd\providecommand
8682 }
8683 \def\DeclareTextSymbol#1#2#3{%
8684 \@dec@text@cmd\chardef#1{#2}#3\relax
8685 }
8686 \def\@dec@text@cmd#1#2#3{%
8687 \expandafter\def\expandafter#2%
8688 \expandafter{%
8689 \csname#3-cmd\expandafter\endcsname
8690 \expandafter#2%
8691 \csname#3\string#2\endcsname
8692 }%
8693 % \let\@ifdefinable\rc@ifdefinable
8694 \expandafter#1\csname#3\string#2\endcsname
8695 }
8696 \def\@current@cmd#1{%
8697 \ifx\protect\@typeset@protect\else
8698 \noexpand#1\expandafter\@gobble
8699 \fi
8700 }
8701 \def\@changed@cmd#1#2{%
8702 \ifx\protect\@typeset@protect
8703 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8704 \expandafter\ifx\csname ?\string#1\endcsname\relax
8705 \expandafter\def\csname ?\string#1\endcsname{%
8706 \@changed@x@err{#1}%
8707 }%
8708 \fi
8709 \global\expandafter\let
8710 \csname\cf@encoding\string#1\expandafter\endcsname
8711 \csname ?\string#1\endcsname
8712 \fi
8713 \csname\cf@encoding\string#1%
8714 \expandafter\endcsname
8715 \else
8716 \noexpand#1%
8717 \fi
8718 }

```

```

8719 \def\@changed@x@err#1{%
8720   \errhelp{Your command will be ignored, type <return> to proceed}%
8721   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8722 \def\DeclareTextCommandDefault#1{%
8723   \DeclareTextCommand#1?%
8724 }
8725 \def\ProvideTextCommandDefault#1{%
8726   \ProvideTextCommand#1?%
8727 }
8728 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8729 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8730 \def\DeclareTextAccent#1#2#3{%
8731   \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
8732 }
8733 \def\DeclareTextCompositeCommand#1#2#3#4{%
8734   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8735   \edef\reserved@b{\string##1}%
8736   \edef\reserved@c{%
8737     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8738   \ifx\reserved@b\reserved@c
8739     \expandafter\expandafter\expandafter\ifx
8740       \expandafter\@car\reserved@a\relax\relax\@nil
8741       \@text@composite
8742     \else
8743       \edef\reserved@b##1{%
8744         \def\expandafter\noexpand
8745           \csname#2\string#1\endcsname###1{%
8746             \noexpand\@text@composite
8747               \expandafter\noexpand\csname#2\string#1\endcsname
8748                 ###1\noexpand\@empty\noexpand\@text@composite
8749                 {##1}%
8750             }%
8751         }%
8752       \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8753     \fi
8754     \expandafter\def\csname\expandafter\string\csname
8755       #2\endcsname\string#1-\string#3\endcsname{#4}
8756   \else
8757     \errhelp{Your command will be ignored, type <return> to proceed}%
8758     \errmessage{\string\DeclareTextCompositeCommand\space used on
8759       inappropriate command \protect#1}
8760   \fi
8761 }
8762 \def\@text@composite#1#2#3\@text@composite{%
8763   \expandafter\@text@composite@x
8764     \csname\string#1-\string#2\endcsname
8765 }
8766 \def\@text@composite@x#1#2{%
8767   \ifx#1\relax
8768     #2%
8769   \else
8770     #1%
8771   \fi
8772 }
8773 %
8774 \def\@strip@args#1:#2-#3\@strip@args{#2}
8775 \def\DeclareTextComposite#1#2#3#4{%
8776   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8777   \bgroup
8778     \lccode`\@=#4%
8779     \lowercase{%
8780   \egroup
8781     \reserved@a @%

```

```

8782 }%
8783 }
8784 %
8785 \def\UseTextSymbol#1#2{#2}
8786 \def\UseTextAccent#1#2#3{}
8787 \def\@use@text@encoding#1{}
8788 \def\DeclareTextSymbolDefault#1#2{%
8789 \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8790 }
8791 \def\DeclareTextAccentDefault#1#2{%
8792 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8793 }
8794 \def\cf@encoding{OT1}

```

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

```

8795 \DeclareTextAccent{"}{OT1}{127}
8796 \DeclareTextAccent{'}{OT1}{19}
8797 \DeclareTextAccent{^}{OT1}{94}
8798 \DeclareTextAccent{\`}{OT1}{18}
8799 \DeclareTextAccent{\~}{OT1}{126}

```

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

```

8800 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8801 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
8802 \DeclareTextSymbol{\textquoteleft}{OT1}{`\'}
8803 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
8804 \DeclareTextSymbol{\i}{OT1}{16}
8805 \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`.

```

8806 \ifx\scriptsize\@undefined
8807 \let\scriptsize\sevenrm
8808 \fi

```

And a few more “dummy” definitions.

```

8809 \def\languagename{english}%
8810 \let\bbl@opt@shorthands\@nnil
8811 \def\bbl@ifshorthand#1#2#3{#2}%
8812 \let\bbl@language@opts\@empty
8813 \let\bbl@ensureinfo\@gobble
8814 \let\bbl@provide@locale\relax
8815 \ifx\babeloptionstrings\@undefined
8816 \let\bbl@opt@strings\@nnil
8817 \else
8818 \let\bbl@opt@strings\babeloptionstrings
8819 \fi
8820 \def\BabelStringsDefault{generic}
8821 \def\bbl@tempa{normal}
8822 \ifx\babeloptionmath\bbl@tempa
8823 \def\bbl@mathnormal{\noexpand\textormath}
8824 \fi
8825 \def\AfterBabelLanguage#1#2{}
8826 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
8827 \let\bbl@afterlang\relax
8828 \def\bbl@opt@safe{BR}
8829 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
8830 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
8831 \expandafter\newif\csname ifbbl@single\endcsname
8832 \chardef\bbl@bidimode\z@
8833 <</Emulate LaTeX>>

```

A proxy file:

```
8834 <*plain>
8835 \input babel.def
8836 </plain>
```

15 Acknowledgements

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

References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national \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 \TeX* , *TUGboat* 9 (1988) #1, p. 70–72.
- [11] Joachim Schrod, *International \LaTeX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using \LaTeX* , Springer, 2002, p. 301–373.
- [13] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).