

Babel

Code

Version 3.92.20074
2023/07/19

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Localization and
internationalization

Unicode

T_EX

pdfT_EX

LuaT_EX

XeT_EX

Contents

1	Identification and loading of required files	3
2	locale directory	3
3	Tools	3
3.1	Multiple languages	7
3.2	The Package File (<code>\LaTeX</code> , <code>babel.sty</code>)	8
3.3	<code>base</code>	9
3.4	<code>key=value</code> options and other general option	10
3.5	Conditional loading of shorthands	11
3.6	Interlude for Plain	13
4	Multiple languages	13
4.1	Selecting the language	15
4.2	Errors	23
4.3	Hooks	25
4.4	Setting up language files	27
4.5	Shorthands	29
4.6	Language attributes	38
4.7	Support for saving macro definitions	40
4.8	Short tags	41
4.9	Hyphens	42
4.10	Multiencoding strings	43
4.11	Macros common to a number of languages	49
4.12	Making glyphs available	49
4.12.1	Quotation marks	50
4.12.2	Letters	51
4.12.3	Shorthands for quotation marks	52
4.12.4	Umlauts and tremas	53
4.13	Layout	54
4.14	Load engine specific macros	54
4.15	Creating and modifying languages	55
5	Adjusting the Babel bahavior	77
5.1	Cross referencing macros	79
5.2	Marks	82
5.3	Preventing clashes with other packages	83
5.3.1	<code>ifthen</code>	83
5.3.2	<code>varioref</code>	83
5.3.3	<code>hhline</code>	84
5.4	Encoding and fonts	84
5.5	Basic bidi support	86
5.6	Local Language Configuration	89
5.7	Language options	89
6	The kernel of Babel (<code>babel.def</code>, <code>common</code>)	93
7	Loading hyphenation patterns	93
8	Font handling with <code>fontspec</code>	97
9	Hooks for XeTeX and LuaTeX	101
9.1	XeTeX	101
9.2	Layout	102
9.3	8-bit TeX	104
9.4	LuaTeX	105
9.5	Southeast Asian scripts	111
9.6	CJK line breaking	112

9.7	Arabic justification	114
9.8	Common stuff	118
9.9	Automatic fonts and ids switching	119
9.10	Bidi	125
9.11	Layout	127
9.12	Lua: transforms	134
9.13	Lua: Auto bidi with basic and basic-r	143
10	Data for CJK	153
11	The ‘nil’ language	153
12	Calendars	155
12.1	Islamic	155
12.2	Hebrew	156
12.3	Persian	160
12.4	Coptic and Ethiopic	161
12.5	Buddhist	162
13	Support for Plain T_EX (plain.def)	162
13.1	Not renaming hyphen.tex	162
13.2	Emulating some L ^A T _E X features	163
13.3	General tools	163
13.4	Encoding related macros	167
14	Acknowledgements	169

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

1 Identification and loading of required files

Code documentation is still under revision.

The babel package after unpacking consists of the following files:

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

babel.def is loaded by Plain.

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

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

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

There some additional tex, def and lua files

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

2 locale directory

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

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

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

3 Tools

```
1 <<version=3.92.20074>>
2 <<date=2023/07/19>>
```

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

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

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

```

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

```

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

```

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

```

\bbl@afterelse Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the `\else` and `\fi` parts of an `\if`-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\csname#1\endcsname}%
41 \def\bbl@exp@ue#1]{%
42   \unexpanded\expandafter\expandafter\expandafter{\csname#1\endcsname}}%

```

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

```

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

```

\bbl@ifunset To check if a macro is defined, we create a new macro, which does the same as `\@ifundefined`. However, in an 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{\bb@ifblank{\@nameuse{#1}}}{#3}{#2}}}

```

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

```

81 \def\bb@forkv#1#2{%
82   \def\bb@kvcmd##1##2##3{#2}%
83   \bb@kvnext#1,\@nil,}
84 \def\bb@kvnext#1,{%
85   \ifx\@nil#1\relax\else
86     \bb@ifblank{#1}{\bb@forkv@eq#1=\@empty=\@nil{#1}}%
87     \expandafter\bb@kvnext
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\bbl@nil##2%
105 \toks@{\expandafter{\the\toks@##1}%
106 \else
107 \toks@{\expandafter{\the\toks@##1#3}%
108 \bbl@afterfi
109 \bbl@replace@aux##2#2%
110 \fi}%
111 \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
112 \edef#1{\the\toks@}}

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

3.1 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)>> \equiv

```



```

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#1.1df-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 \TeX forgets about the first loading. After a subset of `babel.def` has been loaded (the old `switch.def`) and `\AfterBabelLanguage` defined, it exits.

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

```

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

```

3.4 key=value options and other general option

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

```

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

```

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

```

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

```

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

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

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

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

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

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

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

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

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

3.5 Conditional loading of shorthands

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

3.6 Interlude for Plain

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

```

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

```

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

```

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

```

4 Multiple languages

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

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

```

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

```

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

```

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

```

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

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

```

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

```

```

471 \edef\bbl@tempd{\noexpand\ifundefined{\noexpand\bbl@tempe#1}}%
472 \bbl@tempd
473 {\lowercase\expandafter{\bbl@tempd}%
474 {\uppercase\expandafter{\bbl@tempd}%
475 \@empty
476 {\edef\bbl@tempd{\def\noexpand#1{#1}}%
477 \uppercase\expandafter{\bbl@tempd}}}%
478 {\edef\bbl@tempd{\def\noexpand#1{#1}}%
479 \lowercase\expandafter{\bbl@tempd}}}%
480 \@empty
481 \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
482 \bbl@tempd
483 \bbl@exp{\bbl@usehooks{language}{\{language\}{#1}}}%
484 \def\bbl@iflanguage#1{%
485 \ifundefined{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\@#5{%
494 \let\bbl@bcp\relax
495 \lowercase{\def\bbl@tempa{#1}}%
496 \ifx\@empty#2%
497 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
498 \else\ifx\@empty#3%
499 \bbl@bcpcase#2\@empty\@empty\@#5\bbl@tempb
500 \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
501 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
502 {}%
503 \ifx\bbl@bcp\relax
504 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
505 \fi
506 \else
507 \bbl@bcpcase#2\@empty\@empty\@#5\bbl@tempb
508 \bbl@bcpcase#3\@empty\@empty\@#5\bbl@tempc
509 \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
510 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
511 {}%
512 \ifx\bbl@bcp\relax
513 \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
514 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
515 {}%
516 \fi
517 \ifx\bbl@bcp\relax
518 \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
519 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
520 {}%
521 \fi
522 \ifx\bbl@bcp\relax
523 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
524 \fi
525 \fi\fi}
526 \let\bbl@initload\relax
527 (-core)

```

```

528 \def\babelprovide@locale{%
529   \ifx\babelprovide@undefined
530     \babelerror{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\localename{\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}{\localename}%
552       \fi
553     \fi
554   \fi
555   \expandafter\ifx\csname date\language\endcsname\relax
556     \IfFileExists{babel-\language.tex}%
557     {\babel@exp{\babelprovide[\babel@autoload@options]{\language}}}%
558     {}%
559   \fi}
560 (+core)

```

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

```

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

```

4.1 Selecting the language

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

```

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

```

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

```

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

```

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

```

573 \let\xstring\string

```


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```

601 \advance\count@\@ne
602 \bbl@csarg\chardef{id@@\language}\count@
603 \edef\bbl@id@last{\the\count@}%
604 \ifcase\bbl@engine\or
605 \directlua{
606   Babel = Babel or {}
607   Babel.locale_props = Babel.locale_props or {}
608   Babel.locale_props[\bbl@id@last] = {}
609   Babel.locale_props[\bbl@id@last].name = '\language'
610 }%
611 \fi}%
612 {}%
613 \chardef\localeid\bbl@c{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 of `\language`. To catch either form a trick is used, but unfortunately as a side effect the catcodes of letters in `\language` are messed up. This is a bug, but preserved for backwards compatibility. The list of auxiliary files can be extended by redefining `\BabelContentsFiles`, but make sure they are loaded inside a group (as `aux`, `toc`, `lof`, and `lot` do) or the last language of the document will remain active afterwards.

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

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

```

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

```

```

649 \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
650 \bbl@savelastskip
651 \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
652 \bbl@restorelastskip
653 \fi
654 \bbl@usehooks{write}}}%
655 \fi
656 \fi}
657 %
658 \let\bbl@restorelastskip\relax
659 \let\bbl@savelastskip\relax
660 %
661 \newif\ifbbl@bcpallowed
662 \bbl@bcpallowedfalse
663 \def\select@language#1{% from set@, babel@aux
664 \ifx\bbl@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 `\curname` primitive. Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of `\selectlanguage`, and calling these macros.

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

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

```

683 \newif\ifbbl@usedategroup
684 \let\bbl@savextras\@empty
685 \def\bbl@switch#1{% from select@, foreign@
686 % make sure there is info for the language if so requested
687 \bbl@ensureinfo{#1}%
688 % restore
689 \originalTeX
690 \expandafter\def\expandafter\originalTeX\expandafter{%
691 \curname noextras#1\endcurname
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@hymap>4\else
728 \csname\language\name @bbl@hyphenmap\endcsname
729 \fi
730 \chardef\bbl@opt@hyphenmap\z@
731 \else
732 \ifnum\bbl@hymap>\bbl@opt@hyphenmap\else
733 \csname\language\name @bbl@hyphenmap\endcsname
734 \fi
735 \fi
736 \let\bbl@hymap\@cclv
737 % hyphenation - select rules
738 \ifnum\csname l@\language\endcsname=\l@unhyphenated
739 \edef\bbl@tempa{u}%
740 \else
741 \edef\bbl@tempa{\bbl@cl{\lnbrk}}%
742 \fi
743 % linebreaking - handle u, e, k (v in the future)
744 \bbl@xin@{/u}{/\bbl@tempa}%
745 \ifin@ \else \bbl@xin@{/e}{/\bbl@tempa} \fi % elongated forms
746 \ifin@ \else \bbl@xin@{/k}{/\bbl@tempa} \fi % only kashida
747 \ifin@ \else \bbl@xin@{/p}{/\bbl@tempa} \fi % padding (eg, Tibetan)
748 \ifin@ \else \bbl@xin@{/v}{/\bbl@tempa} \fi % variable font
749 \ifin@
750 % unhyphenated/kashida/elongated/padding = allow stretching
751 \language\l@unhyphenated
752 \babel@savevariable\emergencystretch
753 \emergencystretch\maxdimen
754 \babel@savevariable\hbadness
755 \hbadness\@M
756 \else
757 % other = select patterns
758 \bbl@patterns{#1}%
759 \fi

```

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

4.2 Errors

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

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

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

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

```

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

```



```

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

```

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

```

949 \ifx\directlua\@undefined\else
950   \ifx\bbl@luapatterns\@undefined
951     \input luababel.def
952   \fi
953 \fi
954 \bbl@trace{Compatibility with language.def}
955 \ifx\bbl@languages\@undefined
956   \ifx\directlua\@undefined
957     \openin1 = language.def % TODO. Remove hardcoded number
958     \ifeof1
959       \closein1
960       \message{I couldn't find the file language.def}
961     \else
962       \closein1
963       \begingroup
964         \def\addlanguage#1#2#3#4#5{%
965           \expandafter\ifx\csname lang@#1\endcsname\relax\else
966             \global\expandafter\let\csname l@#1\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   \beginingroup
989     \lccode`~=#2\relax
990     \lowercase{\endgroup#1~}}

```

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

```

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

```

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

```

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

```

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

```

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

```

4.3 Hooks

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

```

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

```

```

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

```

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

```

1032 \def\bbl@evargs{,% <- don't delete this comma
1033 everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1034 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,language=2,begindocument=1}
1038 \ifx\NewHook\@undefined\else % Test for Plain (?)
1039 \def\bbl@tempa#1=#2\@{ \NewHook{babel/#1}}
1040 \bbl@foreach\bbl@evargs{\bbl@tempa#1\@{
1041 \fi

```

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

```

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

```

```

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

```

4.4 Setting up language files

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

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

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

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

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

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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\language{#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     \begingroup
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   \begingroup
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@files@w
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

`\active@prefix⟨char⟩\normal@char⟨char⟩`

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

```

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\bbl@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\bbl@restoreactive@gobble}%
1320   {\def\bbl@restoreactive#1{%
1321     \bbl@exp{%
1322       \\\AfterBabelLanguage\\\CurrentOption
1323       {\catcode`#1=\the\catcode`#1\relax}%
1324       \\\AtEndOfPackage
1325       {\catcode`#1=\the\catcode`#1\relax}}}%
1326   \AtEndOfPackage{\let\bbl@restoreactive@gobble}}

```

\bbl@sh@select This command helps the shorthand supporting macros to select how to proceed. Note that this macro needs to be expandable as do all the shorthand macros in order for them to work in expansion-only environments such as the argument of `\hyphenation`.

This macro expects the name of a group of shorthands in its first argument and a shorthand character in its second argument. It will expand to either `\bbl@firstcs` or `\bbl@scndcs`. Hence two more arguments need to follow it.

```

1327 \def\bbl@sh@select#1#2{%
1328   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1329     \bbl@afterelse\bbl@scndcs
1330   \else
1331     \bbl@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 `\ifincsn` is available. If there is, the expansion will be more robust.

```

1333 \begingroup
1334 \bbl@ifunset{ifincsn}% 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   \ifincsn
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@activetrue`), something like “`”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 name of the level or group is stored in a macro. The default is to have a user group; use language `\language@group` group ‘english’ and have a system group called ‘system’.

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

`\@notshorthand`

```

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

```

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

```

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

```

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

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

```

1481 \def\bbl@switch@sh#1#2{%
1482   \ifx#2\@nnil\else
1483     \bbl@ifunset{\bbl@active@\string#2}%
1484     {\bbl@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       \bbl@ifunset{\bbl@shdef@\string#2}%
1493       {}%
1494       {\bbl@withactive{\expandafter\let\expandafter}%2%
1495         \csname bbl@shdef@\string#2\endcsname
1496         \bbl@csarg\let{\shdef@\string#2}\relax}%
1497       \ifcase\bbl@activated\or
1498       \bbl@activate{#2}%
1499       \else
1500       \bbl@deactivate{#2}%
1501       \fi
1502       \or
1503       \bbl@ifunset{\bbl@shdef@\string#2}%
1504       {\bbl@withactive{\bbl@csarg\let{\shdef@\string#2}}#2}%
1505       {}%
1506       \csname bbl@oricat@\string#2\endcsname
1507       \csname bbl@oridef@\string#2\endcsname
1508       \fi}%
1509   \bbl@afterfi\bbl@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\bbl@putsh}
1512 \def\bbl@putsh#1{%
1513   \bbl@ifunset{\bbl@active@\string#1}%
1514   {\bbl@putsh@i#1\@empty\@nnil}%
1515   {\csname bbl@active@\string#1\endcsname}}
1516 \def\bbl@putsh@i#1#2\@nnil{%
1517   \csname\language@group @sh@\string#1@%
1518   \ifx\@empty#2\else\string#2@\fi\endcsname}
1519 %
1520 \ifx\bbl@opt@shorthands\@nnil\else
1521   \let\bbl@s@initiate@active@char\initiate@active@char
1522   \def\initiate@active@char#1{%
1523     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1524   \let\bbl@s@switch@sh\bbl@switch@sh
1525   \def\bbl@switch@sh#1#2{%
1526     \ifx#2\@nnil\else
1527     \bbl@afterfi
1528     \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1529     \fi}
1530   \let\bbl@s@activate\bbl@activate
1531   \def\bbl@activate#1{%
1532     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1533   \let\bbl@s@deactivate\bbl@deactivate
1534   \def\bbl@deactivate#1{%
1535     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1536 \fi

```

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

or off.

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

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

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

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

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

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

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

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

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

4.6 Language attributes

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

1593 \def\bbl@declare@ttribute#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\bbl@ifknown@ttrib#1#2{%
1612   \let\bbl@tempa\@secondoftwo
1613   \bbl@loopx\bbl@tempb{#2}{%
1614     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{, #1,}%
1615     \ifin@
1616       \let\bbl@tempa\@firstoftwo
1617     \else
1618       \fi}%
1619   \bbl@tempa}

```

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

```

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

```

4.7 Support for saving macro definitions

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

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

```

1629 \bbl@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\bbl@tempa{, #1,}% Clumsy, for Plain
1635   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1636     \expandafter{\expandafter,\bbl@savedextras,}%
1637     \expandafter\in@\bbl@tempa
1638     \ifin@\else
1639       \bbl@add\bbl@savedextras{, #1,}%
1640       \bbl@carg\let\babel@number\babel@savecnt\#1\relax
1641       \toks@\expandafter{\originalTeX\let#1=}
1642       \bbl@exp{%
1643         \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1644       \advance\babel@savecnt\@ne

```

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

```

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

```

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

```

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

```

4.8 Short tags

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

```

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

```

```

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

```

4.9 Hyphens

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

```

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

```

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

```

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

```

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

```

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

```

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

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

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

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

The following macro inserts the hyphen char.

```
1743 \def\bbl@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 \bbl@hy@nobreak is redundant.

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

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

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

4.10 Multiencoding strings

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

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

```
1764 \bbl@trace{Multiencoding strings}
1765 \def\bbl@tglobal#1{\global\let#1#1}
```

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

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

and starts over (and similarly when lowercasing).

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

```

1768 {\def\bbbl@patchucllc{% TODO. Delete. Doesn't work any more.
1769   \global\let\bbbl@patchucllc\relax
1770   \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbbl@ucllc}}%
1771   \gdef\bbbl@ucllc##1{%
1772     \let\bbbl@encoded\bbbl@encoded@ucllc
1773     \bbbl@ifunset{\language @bbbl@ucllc}% and resumes it
1774     {##1}%
1775     {\let\bbbl@tempa##1\relax % Used by LANG@bbbl@ucllc
1776       \csname\language @bbbl@ucllc\endcsname}%
1777     {\bbbl@tolower\@empty}{\bbbl@toupper\@empty}}}%
1778   \gdef\bbbl@tolower{\csname\language @bbbl@lc\endcsname}%
1779   \gdef\bbbl@toupper{\csname\language @bbbl@uc\endcsname}}}%

1780 <<(*More package options)>> ≡
1781 \DeclareOption{nocase}{}
1782 <</More package options>>

```

The following package options control the behavior of \SetString.

```

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

```

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

```

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

```

```

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

```

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

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

```

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

```

```

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

```

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

```

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

```

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

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```



```

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

```

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

```

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

```

The following package options control the behavior of hyphenation mapping.

```

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

```

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

```

2003 \AtEndOfPackage{%
2004 \ifx\bbl@opt@hyphenmap\undefined
2005 \bbl@xin@{,}{\bbl@language@opts}%
2006 \chardef\bbl@opt@hyphenmap\ifin4\else\@ne\fi
2007 \fi}

```

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

```

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

```

```

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

```

4.11 Macros common to a number of languages

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

```

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

```

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

```

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

```

4.12 Making glyphs available

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

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

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

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

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

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

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

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

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

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

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

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

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

```

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

```

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

```

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

```

4.12.2 Letters

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

```

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

```

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

```

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

```

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

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

```

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

```

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

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

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

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

4.12.3 Shorthands for quotation marks

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

\glq The ‘german’ single quotes.

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

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

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

\glqq The ‘german’ double quotes.

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

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

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

\flq The ‘french’ single guillemets.

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

\flqq The ‘french’ double guillemets.

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

4.12.4 Umlauts and tremas

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

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

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

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

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

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

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

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

For all vowels we declare `\` to be a composite command which uses `\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).

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

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

4.13 Layout

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

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

4.14 Load engine specific macros

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

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

```

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

```

4.15 Creating and modifying languages

Continue with \TeX only.

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

```

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

```



```

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

```

```

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

```

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

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```



```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```



```

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

```

Transforms.

```

3318 \let\bbbl@release@transforms\@empty
3319 \bbbl@csarg\let{inikv@transforms.prehyphenation}\bbbl@inikv
3320 \bbbl@csarg\let{inikv@transforms.posthyphenation}\bbbl@inikv
3321 \def\bbbl@transforms@aux#1#2#3#4,#5\relax{%
3322   #1[#2]{#3}{#4}{#5}}
3323 \begingroup % A hack. TODO. Don't require an specific order
3324 \catcode\%=12
3325 \catcode\&=14
3326 \gdef\bbbl@transforms#1#2#3{%&
3327   \directlua{
3328     local str = [=[#2]=]
3329     str = str:gsub('%.%d+%.%d+$', '')
3330     token.set_macro('babeltempa', str)
3331   }&
3332   \def\babeltempc{ }&
3333   \bbbl@xin@{,\babeltempa,}{,\bbbl@KVP@transforms,}&
3334   \ifin@ \else
3335     \bbbl@xin@{: \babeltempa,}{,\bbbl@KVP@transforms,}&
3336   \fi
3337   \ifin@
3338     \bbbl@foreach\bbbl@KVP@transforms{&
3339     \bbbl@xin@{: \babeltempa,}{,##1,}&

```

```

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

```

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

```

3373 \def\bbl@provide@lsys#1{%
3374     \bbl@ifunset{bbl@lname@#1}%
3375     {\bbl@load@info{#1}}%
3376     }%
3377 \bbl@csarg\let{lsys@#1}\@empty
3378 \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3379 \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3380 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3381 \bbl@ifunset{bbl@lname@#1}{%
3382     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3383 \ifcase\bbl@engine\or\or
3384     \bbl@ifunset{bbl@prehc@#1}{%
3385         {\bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3386         }%
3387         {\ifx\bbl@xenohyph\undefined
3388             \global\let\bbl@xenohyph\bbl@xenohyph@d
3389             \ifx\AtBeginDocument\@notprerr
3390                 \expandafter\@secondoftwo % to execute right now
3391             \fi
3392             \AtBeginDocument{%
3393                 \bbl@patchfont{\bbl@xenohyph}%
3394                 \expandafter\select@language\expandafter{\language}%
3395             \fi}}%
3396     \fi
3397     \bbl@csarg\bbl@to@global{lsys@#1}}
3398 \def\bbl@xenohyph@d{%
3399     \bbl@ifset{bbl@prehc@language}%

```

```

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

```

3418 \def\bbl@load@info#1{%
3419   \def\BabelBeforeIni##1##2{%
3420     \begingroup
3421       \bbl@read@ini{##1}0%
3422       \endinput           % babel- .tex may contain only preamble's
3423       \endgroup}%        boxed, to avoid extra spaces:
3424   {\bbl@input@texini{#1}}

```

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

```

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

```

```
3455 \bbl@tempa}
```

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

```
3456 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}%
3457 \ifx\#1% % \ before, in case #1 is multiletter
3458 \bbl@exp{%
3459 \def\#1\bbl@tempa###1{%
3460 \<ifcase>###1\space\the\toks@\<else>\<\ctrerr\<fi>}}%
3461 \else
3462 \toks@\expandafter{\the\toks@\or #1}%
3463 \expandafter\bbl@buildifcase
3464 \fi}
```

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

```
3465 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1\language}\{#2}}
3466 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3467 \newcommand\localecounter[2]{%
3468 \expandafter\bbl@localecntr
3469 \expandafter{\number\csname c@#2\endcsname}\{#1}}
3470 \def\bbl@alphnumeral#1#2{%
3471 \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3472 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3473 \ifcase\car#8\@nil\or % Currenty <10000, but prepared for bigger
3474 \bbl@alphnumeral@ii{#9}000000#1\or
3475 \bbl@alphnumeral@ii{#9}000000#1#2\or
3476 \bbl@alphnumeral@ii{#9}000000#1#2#3\or
3477 \bbl@alphnumeral@ii{#9}000000#1#2#3#4\else
3478 \bbl@alphnum@invalid{>9999}%
3479 \fi}
3480 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3481 \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8\language}%
3482 {\bbl@cs{cntr@#1.4\language}\{#5}}
3483 {\bbl@cs{cntr@#1.3\language}\{#6}}
3484 {\bbl@cs{cntr@#1.2\language}\{#7}}
3485 {\bbl@cs{cntr@#1.1\language}\{#8}}
3486 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3487 \bbl@ifunset{bbl@cntr@#1.S.321\language}\{#6}}
3488 {\bbl@cs{cntr@#1.S.321\language}\{#6}}
3489 \fi}%
3490 {\bbl@cs{cntr@#1.F.\number#5#6#7#8\language}}
3491 \def\bbl@alphnum@invalid#1{%
3492 \bbl@error{Alphabetic numeral too large (#1)}%
3493 {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.

```
3494 \def\bbl@localeinfo#1#2{%
3495 \bbl@ifunset{bbl@info@#2}\{#1}}
3496 {\bbl@ifunset{bbl@csname bbl@info@#2\endcsname @\language}\{#1}}
3497 {\bbl@cs{csname bbl@info@#2\endcsname @\language}}
3498 \newcommand\localeinfo[1]{%
3499 \ifx*#1\@empty % TODO. A bit hackish to make it expandable.
3500 \bbl@afterelse\bbl@localeinfo}%
3501 \else
3502 \bbl@localeinfo
3503 {\bbl@error{I've found no info for the current locale.\%
3504 The corresponding ini file has not been loaded\%
3505 Perhaps it doesn't exist}%
3506 {See the manual for details.}}%
3507 {#1}}
```

```

3508 \fi}
3509 % \@namedef{bbl@info@name.locale}{lcname}
3510 \@namedef{bbl@info@tag.ini}{lini}
3511 \@namedef{bbl@info@name.english}{elname}
3512 \@namedef{bbl@info@name.opentype}{lname}
3513 \@namedef{bbl@info@tag.bcp47}{tbc}
3514 \@namedef{bbl@info@language.tag.bcp47}{lbcp}
3515 \@namedef{bbl@info@tag.opentype}{lotf}
3516 \@namedef{bbl@info@script.name}{esname}
3517 \@namedef{bbl@info@script.name.opentype}{sname}
3518 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3519 \@namedef{bbl@info@script.tag.opentype}{sotf}
3520 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3521 \@namedef{bbl@info@variant.tag.bcp47}{vbc}
3522 \@namedef{bbl@info@extension.t.tag.bcp47}{extt}
3523 \@namedef{bbl@info@extension.u.tag.bcp47}{extu}
3524 \@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.

```

3525 \providecommand\BCPdata{}
3526 \ifx\renewcommand\@undefined\else % For plain. TODO. It's a quick fix
3527 \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty}
3528 \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3529 \nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3530 {\bbl@bcpdata@ii{#6}\bbl@main@language}%
3531 {\bbl@bcpdata@ii{#1#2#3#4#5#6}\language}%}
3532 \def\bbl@bcpdata@ii#1#2{%
3533 \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3534 {\bbl@error{Unknown field '#1' in \string\BCPdata.\%
3535 Perhaps you misspelled it.}%
3536 {See the manual for details.}}%
3537 {\bbl@ifunset{bbl@csname bbl@info@#1.tag.bcp47\endcsname @#2}{}%
3538 {\bbl@cs{csname bbl@info@#1.tag.bcp47\endcsname @#2}}}%
3539 \fi
3540 % Still somewhat hackish. WIP.
3541 \@namedef{bbl@info@casing.tag.bcp47}{casing}
3542 \newcommand\BabelUppercaseMapping[3]{%
3543 \let\bbl@temp\language
3544 \edef\language{#1}%
3545 \DeclareUppercaseMapping[\BCPdata{casing}]{#2}{#3}%
3546 \let\language\bbl@temp}
3547 \newcommand\BabelLowercaseMapping[3]{%
3548 \let\bbl@temp\language
3549 \edef\language{#1}%
3550 \DeclareLowercaseMapping[\BCPdata{casing}]{#2}{#3}%
3551 \let\language\bbl@temp}

```

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

```

3552 <<*More package options>> \equiv
3553 \DeclareOption{ensureinfo=off}{}
3554 <</More package options>>
3555 \let\bbl@ensureinfo\@gobble
3556 \newcommand\BabelEnsureInfo{%
3557 \ifx\InputIfFileExists\@undefined\else
3558 \def\bbl@ensureinfo##1{%
3559 \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3560 \fi
3561 \bbl@foreach\bbl@loaded{%
3562 \let\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files
3563 \def\language{##1}%
3564 \bbl@ensureinfo{##1}}}%
3565 \ifpackagewith{babel}{ensureinfo=off}{}%

```

```

3566 {\AtEndOfPackage{% Test for plain.
3567 \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`.

```

3568 \newcommand\getlocaleproperty{%
3569 \ifstar\bbl@getproperty@s\bbl@getproperty@x}
3570 \def\bbl@getproperty@s#1#2#3{%
3571 \let#1\relax
3572 \def\bbl@elt##1##2##3{%
3573 \bbl@ifsamestring{##1/##2}{##3}%
3574 {\providecommand#1{##3}%
3575 \def\bbl@elt###1###2###3{}}}%
3576 {}}%
3577 \bbl@cs{inidata@#2}}%
3578 \def\bbl@getproperty@x#1#2#3{%
3579 \bbl@getproperty@s{#1}{#2}{#3}%
3580 \ifx#1\relax
3581 \bbl@error
3582 {Unknown key for locale '#2':\%
3583 #3\%
3584 \string#1 will be set to \relax}%
3585 {Perhaps you misspelled it.}%
3586 \fi}
3587 \let\bbl@ini@loaded\@empty
3588 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

5 Adjusting the Babel behavior

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

```

3589 \newcommand\babeladjust[1]{% TODO. Error handling.
3590 \bbl@forkv{#1}{%
3591 \bbl@ifunset\bbl@ADJ@##1@##2}%
3592 {\bbl@cs{ADJ@##1}{##2}}%
3593 {\bbl@cs{ADJ@##1@##2}}}%
3594 %
3595 \def\bbl@adjust@lua#1#2{%
3596 \ifvmode
3597 \ifnum\currentgrouplevel=\z@
3598 \directlua{ Babel.#2 }%
3599 \expandafter\expandafter\expandafter\@gobble
3600 \fi
3601 \fi
3602 {\bbl@error % The error is gobbled if everything went ok.
3603 {Currently, #1 related features can be adjusted only\%
3604 in the main vertical list.}%
3605 {Maybe things change in the future, but this is what it is.}}}
3606 \@namedef\bbl@ADJ@bidi.mirroring@on{%
3607 \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3608 \@namedef\bbl@ADJ@bidi.mirroring@off{%
3609 \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3610 \@namedef\bbl@ADJ@bidi.text@on{%
3611 \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3612 \@namedef\bbl@ADJ@bidi.text@off{%
3613 \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3614 \@namedef\bbl@ADJ@bidi.math@on{%
3615 \let\bbl@noamsmath\@empty}
3616 \@namedef\bbl@ADJ@bidi.math@off{%
3617 \let\bbl@noamsmath\relax}
3618 \@namedef\bbl@ADJ@bidi.mapdigits@on{%
3619 \bbl@adjust@lua{bidi}{digits_mapped=true}}

```

```

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

```

```

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

```

3712 <<{*More package options}> \equiv
3713 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3714 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3715 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3716 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3717 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3718 <</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.

```

3719 \bbl@trace{Cross referencing macros}
3720 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3721   \def\@newl@bel#1#2#3{%
3722     {\@safe@activestrue
3723       \bbl@ifunset{#1@#2}%
3724       \relax
3725       {\gdef\@multiplelabels{%
3726         \@latex@warning@no@line{There were multiply-defined labels}}%
3727         \@latex@warning@no@line{Label `#2' multiply defined}}%
3728       \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.

```

3729 \CheckCommand*\@testdef[3]{%
3730   \def\reserved@a{#3}%
3731   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3732   \else
3733     \@tempswatru
3734   \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 `\newlabel` 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.

```

3735 \def\@testdef#1#2#3{% TODO. With @samestring?
3736   \@safe@activetrue
3737   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3738   \def\bbl@tempb{#3}%
3739   \@safe@activetrue
3740   \ifx\bbl@tempa\relax
3741   \else
3742     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3743   \fi
3744   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3745   \ifx\bbl@tempa\bbl@tempb
3746   \else
3747     \@tempswatru
3748   \fi}
3749 \fi

```

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

```

3750 \bbl@xin@{R}\bbl@opt@safe
3751 \ifin@
3752   \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3753   \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3754   {\expandafter\strip@prefix\meaning\ref}%
3755 \ifin@
3756   \bbl@redefine\@kernel@ref#1{%
3757     \@safe@activetrue\org@@kernel@ref{#1}\@safe@activetrue}
3758   \bbl@redefine\@kernel@pageref#1{%
3759     \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activetrue}
3760   \bbl@redefine\@kernel@sref#1{%
3761     \@safe@activetrue\org@@kernel@sref{#1}\@safe@activetrue}
3762   \bbl@redefine\@kernel@spageref#1{%
3763     \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activetrue}
3764   \else
3765     \bbl@redefineroobust\ref#1{%
3766       \@safe@activetrue\org@ref{#1}\@safe@activetrue}
3767     \bbl@redefineroobust\pageref#1{%
3768       \@safe@activetrue\org@pageref{#1}\@safe@activetrue}
3769   \fi
3770 \else
3771   \let\org@ref\ref
3772   \let\org@pageref\pageref
3773 \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.

```

3774 \bbl@xin@{B}\bbl@opt@safe
3775 \ifin@
3776 \bbl@redefine\@citex[#1]#2{%
3777   \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
3778   \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.

```

3779 \AtBeginDocument{%
3780   \@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.)

```

3781   \def\@citex[#1][#2]#3{%
3782     \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
3783     \org@citex[#1][#2]{\@tempa}}%
3784   }{}}

```

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

```

3785 \AtBeginDocument{%
3786   \@ifpackageloaded{cite}{%
3787     \def\@citex[#1]#2{%
3788       \@safe@activetrue\org@citex[#1][#2]\@safe@activesfalse}%
3789     }{}}

```

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

```

3790 \bbl@redefine\nocite#1{%
3791   \@safe@activetrue\org@nocite{#1}\@safe@activesfalse}

```

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

```

3792 \bbl@redefine\bibcite{%
3793   \bbl@cite@choice
3794   \bibcite}

```

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

```

3795 \def\bbl@bibcite#1#2{%
3796   \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.

```

3797 \def\bbl@cite@choice{%
3798   \global\let\bibcite\bbl@bibcite
3799   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
3800   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
3801   \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.

```

3802 \AtBeginDocument{\bbl@cite@choice}

```

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

```

3803 \bbl@redefine\@bibitem#1{%
3804   \@safe@activetrue\org@bibitem{#1}\@safe@activesfalse}
3805 \else
3806   \let\org@nocite\nocite
3807   \let\org@citex\@citex
3808   \let\org@bibcite\bibcite
3809   \let\org@bibitem\@bibitem
3810 \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.

```

3811 \bbl@trace{Marks}
3812 \IfBabelLayout{sectioning}
3813   {\ifx\bbl@opt@headfoot\@nnil
3814     \g@addto@macro\@resetactivechars{%
3815       \set@typeset@protect
3816       \expandafter\select@language@x\expandafter{\bbl@main@language}%
3817       \let\protect\noexpand
3818       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3819         \edef\thepage{%
3820           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3821       \fi}%
3822   \fi}
3823 {\ifbbl@single\else
3824   \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
3825   \markright#1{%
3826     \bbl@ifblank{#1}%
3827     {\org@markright{}}%
3828     {\toks@{#1}%
3829       \bbl@exp{%
3830         \\org@markright{\\protect\\foreignlanguage{\language}\language}%
3831         {\\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, \TeX stores the definition in an intermediate macro, so it's not necessary anymore, but it's preserved for older versions.)

```

3832   \ifx\@mkboth\markboth
3833     \def\bbl@tempc{\let\@mkboth\markboth}%
3834   \else
3835     \def\bbl@tempc{%
3836       \fi
3837       \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
3838       \markboth#1#2{%
3839         \protected@edef\bbl@tempb##1{%
3840           \protect\foreignlanguage
3841             {\language}\protect\bbl@restore@actives##1}%
3842         \bbl@ifblank{#1}%
3843         {\toks@{}}%
3844         {\toks@\expandafter{\bbl@tempb{#1}}}%
3845         \bbl@ifblank{#2}%
3846         {\@temptokena{}}%
3847         {\@temptokena\expandafter{\bbl@tempb{#2}}}%

```

```

3848      \bbl@exp{\org@markboth{\the\toks@}{\the\temptokena}}}%
3849      \bbl@tempc
3850    \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.

```

3851 \bbl@trace{Preventing clashes with other packages}
3852 \ifx\org@ref\undefined\else
3853   \bbl@xin@{R}\bbl@opt@safe
3854   \ifin@
3855     \AtBeginDocument{%
3856       \@ifpackageloaded{ifthen}{%
3857         \bbl@redefine@long\ifthenelse#1#2#3{%
3858           \let\bbl@temp@pref\pageref
3859           \let\pageref\org@pageref
3860           \let\bbl@temp@ref\ref
3861           \let\ref\org@ref
3862           \@safe@activestrue
3863           \org@ifthenelse{#1}%
3864             {\let\pageref\bbl@temp@pref
3865              \let\ref\bbl@temp@ref
3866              \@safe@activesfalse
3867              #2}%
3868             {\let\pageref\bbl@temp@pref
3869              \let\ref\bbl@temp@ref
3870              \@safe@activesfalse
3871              #3}%
3872           }%
3873         }{}%
3874       }
3875 \fi

```

5.3.2 varioref

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

```

3876 \AtBeginDocument{%
3877   \@ifpackageloaded{varioref}{%
3878     \bbl@redefine\@vppageref#1[#2]#3{%
3879       \@safe@activestrue
3880       \org@@@vppageref{#1}[#2]{#3}%
3881       \@safe@activesfalse}%
3882     \bbl@redefine\vrefpagenum#1#2{%
3883       \@safe@activestrue

```

```

3884 \org@vrefpagenum{#1}{#2}%
3885 \@safe@activesfalse}%

```

The package `varioref` defines `\Ref` to be a robust command which uppercases the first character of the reference text. In order to be able to do that it needs to access the expandable form of `\ref`. So we employ a little trick here. We redefine the (internal) command `\Ref_` to call `\org@ref` instead of `\ref`. The disadvantage of this solution is that whenever the definition of `\Ref` changes, this definition needs to be updated as well.

```

3886 \expandafter\def\csname Ref \endcsname#1{%
3887 \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3888 }{}%
3889 }
3890 \fi

```

5.3.3 `hhline`

`\hhline` Delaying the activation of the shorthand characters has introduced a problem with the `hhline` package. The reason is that it uses the ‘`’` character which is made active by the french support in `babel`. Therefore we need to *reload* the package when the ‘`’` is an active character. Note that this happens *after* the category code of the `@`-sign has been changed to other, so we need to temporarily change it to letter again.

```

3891 \AtEndOfPackage{%
3892 \AtBeginDocument{%
3893 \ifpackageloaded{hhline}%
3894 {\expandafter\ifx\csname normal@char\string\endcsname\relax
3895 \else
3896 \makeatletter
3897 \def\@currname{hhline}\input{hhline.sty}\makeatother
3898 \fi}%
3899 {}%

```

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

```

3900 \def\substitutefontfamily#1#2#3{%
3901 \lowercase{\immediate\openout15=#1#2.fd\relax}%
3902 \immediate\write15{%
3903 \string\ProvidesFile{#1#2.fd}%
3904 [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3905 \space generated font description file]^J
3906 \string\DeclareFontFamily{#1}{#2}{ }^J
3907 \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{ }^J
3908 \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{ }^J
3909 \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{ }^J
3910 \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{ }^J
3911 \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{ }^J
3912 \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{ }^J
3913 \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{ }^J
3914 \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{ }^J
3915 }%
3916 \closeout15
3917 }
3918 \@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

```
3919 \bbl@trace{Encoding and fonts}
3920 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3921 \newcommand\BabelNonText{TS1,T3,TS3}
3922 \let\org@TeX\TeX
3923 \let\org@LaTeX\LaTeX
3924 \let\ensureascii\@firstofone
3925 \AtBeginDocument{%
3926   \def\@elt#1{,#1,}%
3927   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3928   \let\@elt\relax
3929   \let\bbl@tempb\@empty
3930   \def\bbl@tempc{OT1}%
3931   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3932     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3933   \bbl@foreach\bbl@tempa{%
3934     \bbl@xin@{#1}{\BabelNonASCII}%
3935     \ifin@
3936       \def\bbl@tempb{#1}% Store last non-ascii
3937       \else\bbl@xin@{#1}{\BabelNonText}% Pass
3938       \ifin@\else
3939         \def\bbl@tempc{#1}% Store last ascii
3940       \fi
3941     \fi}%
3942   \ifx\bbl@tempb\@empty\else
3943     \bbl@xin@{\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3944     \ifin@\else
3945       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3946     \fi
3947     \edef\ensureascii#1{%
3948       {\noexpand\fontencoding{\bbl@tempc}\noexpand\selectfont#1}}%
3949     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3950     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3951   \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.

```
3952 \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.

```
3953 \AtBeginDocument{%
3954   \@ifpackageloaded{fontspec}%
3955   {\xdef\latinencoding{%
3956     \ifx\UTFencname\@undefined
3957       EU\ifcase\bbl@engine\or2\or1\fi
3958     \else
3959       \UTFencname
3960     \fi}}%
3961   {\gdef\latinencoding{OT1}%
3962     \ifx\cf@encoding\bbl@t@one
3963       \xdef\latinencoding{\bbl@t@one}%
3964     \else
3965       \def\@elt#1{,#1,}%
3966       \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3967       \let\@elt\relax
3968       \bbl@xin@{,T1,}\bbl@tempa
```

```

3969 \ifin@
3970 \xdef\latinencoding{\bbl@t@one}%
3971 \fi
3972 \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.

```

3973 \DeclareRobustCommand{\latintext}{%
3974 \fontencoding{\latinencoding}\selectfont
3975 \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.

```

3976 \ifx\@undefined\DeclareTextFontCommand
3977 \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3978 \else
3979 \DeclareTextFontCommand{\textlatin}{\latintext}
3980 \fi

```

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

```

3981 \def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}

```

5.5 Basic bidi support

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

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

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

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

```

3982 \bbl@trace{Loading basic (internal) bidi support}
3983 \ifodd\bbl@engine
3984 \else % TODO. Move to txtbabel
3985 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200 % Any xe+lua bidi=
3986 \bbl@error
3987 {The bidi method 'basic' is available only in\\%
3988 luatex. I'll continue with 'bidi=default', so\\%
3989 expect wrong results}%
3990 {See the manual for further details.}%
3991 \let\bbl@beforeforeign\leavevmode
3992 \AtEndOfPackage{%
3993 \EnableBabelHook{babel-bidi}%
3994 \bbl@xebidipar}
3995 \fi\fi
3996 \def\bbl@loadxebidi#1{%
3997 \ifx\RTLfootnotetext\@undefined
3998 \AtEndOfPackage{%

```

```

3999      \EnableBabelHook{babel-bidi}%
4000      \bbl@loadfontspec % bidi needs fontspec
4001      \usepackage#1{bidi}}%
4002  \fi}
4003  \ifnum\bbl@bidimode>200 % Any xe bidi=
4004  \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
4005  \bbl@tentative{bidi=bidi}
4006  \bbl@loadxebidi{}
4007  \or
4008  \bbl@loadxebidi{[rldocument]}
4009  \or
4010  \bbl@loadxebidi{}
4011  \fi
4012  \fi
4013  \fi
4014  % TODO? Separate:
4015  \ifnum\bbl@bidimode=\@ne % Any bidi= except default=1
4016  \let\bbl@beforeforeign\leavevmode
4017  \ifodd\bbl@engine
4018  \newattribute\bbl@attr@dir
4019  \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
4020  \bbl@exp{\output{\bodydir\pagedir\the\output}}
4021  \fi
4022  \AtEndOfPackage{%
4023  \EnableBabelHook{babel-bidi}%
4024  \ifodd\bbl@engine\else
4025  \bbl@xebidipar
4026  \fi}
4027  \fi

```

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

```

4028  \bbl@trace{Macros to switch the text direction}
4029  \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
4030  \def\bbl@rscripts{% TODO. Base on codes ??
4031  ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
4032  Old Hungarian,Lydian,Mandaean,Manichaeen,%
4033  Meroitic Cursive,Meroitic,Old North Arabian,%
4034  Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
4035  Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
4036  Old South Arabian,}%
4037  \def\bbl@provide@dirs#1{%
4038  \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
4039  \ifin@
4040  \global\bbl@csarg\chardef{wdir@#1}\@ne
4041  \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4042  \ifin@
4043  \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
4044  \fi
4045  \else
4046  \global\bbl@csarg\chardef{wdir@#1}\z@
4047  \fi
4048  \ifodd\bbl@engine
4049  \bbl@csarg\ifcase{wdir@#1}%
4050  \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
4051  \or
4052  \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
4053  \or
4054  \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
4055  \fi
4056  \fi}
4057  \def\bbl@switchdir{%
4058  \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}}%

```



```

4059 \bbl@ifunset{\bbl@wdir@\language}\bbl@provide@dirs{\language}}{}%
4060 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}
4061 \def\bbl@setdirs#1{% TODO - math
4062 \ifcase\bbl@select@type % TODO - strictly, not the right test
4063 \bbl@bodydir{#1}%
4064 \bbl@pdir{#1}% <- Must precede \bbl@textdir
4065 \fi
4066 \bbl@textdir{#1}}
4067 % TODO. Only if \bbl@bidimode > 0?:
4068 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4069 \DisableBabelHook{babel-bidi}

```

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

```

4070 \ifodd\bbl@engine % luatex=1
4071 \else % pdftex=0, xetex=2
4072 \newcount\bbl@dirlevel
4073 \chardef\bbl@thetextdir\z@
4074 \chardef\bbl@thepardir\z@
4075 \def\bbl@textdir#1{%
4076 \ifcase#1\relax
4077 \chardef\bbl@thetextdir\z@
4078 \bbl@textdir@i\beginL\endL
4079 \else
4080 \chardef\bbl@thetextdir@ne
4081 \bbl@textdir@i\beginR\endR
4082 \fi}
4083 \def\bbl@textdir@i#1#2{%
4084 \ifhmode
4085 \ifnum\currentgrouplevel>\z@
4086 \ifnum\currentgrouplevel=\bbl@dirlevel
4087 \bbl@error{Multiple bidi settings inside a group}%
4088 {I'll insert a new group, but expect wrong results.}%
4089 \bgroup\aftergroup#2\aftergroup\egroup
4090 \else
4091 \ifcase\currentgrouptype\or % 0 bottom
4092 \aftergroup#2% 1 simple {}
4093 \or
4094 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4095 \or
4096 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4097 \or\or\or % vbox vtop align
4098 \or
4099 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4100 \or\or\or\or\or\or % output math disc insert vcent mathchoice
4101 \or
4102 \aftergroup#2% 14 \begingroup
4103 \else
4104 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4105 \fi
4106 \fi
4107 \bbl@dirlevel\currentgrouplevel
4108 \fi
4109 #1%
4110 \fi}
4111 \def\bbl@pdir#1{\chardef\bbl@thepardir#1\relax}
4112 \let\bbl@bodydir\@gobble
4113 \let\bbl@pagedir\@gobble
4114 \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).

```

4115 \def\bbl@xebidipar{%
4116 \let\bbl@xebidipar\relax

```

```

4117 \TeXeTstate\@ne
4118 \def\bbl@xeverypar{%
4119   \ifcase\bbl@thepardir
4120     \ifcase\bbl@thetextdir\else\beginR\fi
4121   \else
4122     {\setbox\z@\lastbox\beginR\box\z@}%
4123   \fi}%
4124 \let\bbl@severypar\everypar
4125 \newtoks\everypar
4126 \everypar=\bbl@severypar
4127 \bbl@severypar{\bbl@xeverypar\the\everypar}}
4128 \ifnum\bbl@bidimode>200 % Any xe bidi=
4129 \let\bbl@textdir@i\@gobbletwo
4130 \let\bbl@xebidipar\@empty
4131 \AddBabelHook{bidi}{foreign}{%
4132   \def\bbl@tempa{\def\BabelText###1}%
4133   \ifcase\bbl@thetextdir
4134     \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4135   \else
4136     \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4137   \fi}
4138 \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4139 \fi
4140 \fi

```

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

```

4141 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4142 \AtBeginDocument{%
4143   \ifx\pdfstringdefDisableCommands\@undefined\else
4144     \ifx\pdfstringdefDisableCommands\relax\else
4145       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4146     \fi
4147   \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`.

```

4148 \bbl@trace{Local Language Configuration}
4149 \ifx\loadlocalcfg\@undefined
4150   \ifpackagewith{babel}{noconfigs}%
4151     {\let\loadlocalcfg\@gobble}%
4152   {\def\loadlocalcfg#1{%
4153     \InputIfFileExists{#1.cfg}%
4154     {\typeout{*****^J%
4155               * Local config file #1.cfg used^^J%
4156               *}}%
4157     \@empty}}
4158 \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).

```

4159 \bbl@trace{Language options}
4160 \let\bbl@afterlang\relax
4161 \let\BabelModifiers\relax
4162 \let\bbl@loaded\@empty

```

```

4163 \def\bbl@load@language#1{%
4164   \InputIfFileExists{#1.ldf}%
4165   {\edef\bbl@loaded{\CurrentOption
4166     \ifx\bbl@loaded\empty\else,\bbl@loaded\fi}%
4167     \expandafter\let\expandafter\bbl@afterlang
4168     \csname\CurrentOption.ldf-h@k\endcsname
4169     \expandafter\let\expandafter\BabelModifiers
4170     \csname bbl@mod@\CurrentOption\endcsname
4171     \bbl@exp{\AtBeginDocument}%
4172     \bbl@usehooks@lang{\CurrentOption}{begindocument}{\CurrentOption}}}%
4173   {\bbl@error{%
4174     Unknown option '\CurrentOption'. Either you misspelled it\%
4175     or the language definition file \CurrentOption.ldf was not found}%
4176     Valid options are, among others: shorthands=, KeepShorthandsActive,\%
4177     activeacute, activegrave, noconfigs, safe=, main=, math=\%
4178     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.

```

4179 \def\bbl@try@load@lang#1#2#3{%
4180   \IfFileExists{\CurrentOption.ldf}%
4181   {\bbl@load@language{\CurrentOption}}%
4182   {\bbl@load@language{#2#3}}
4183 %
4184 \DeclareOption{hebrew}{%
4185   \input{rlbabel.def}%
4186   \bbl@load@language{hebrew}}
4187 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}}
4188 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}}
4189 \DeclareOption{northersami}{\bbl@try@load@lang{}{samin}}
4190 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}}
4191 \DeclareOption{polutonikogreek}{%
4192   \bbl@try@load@lang{}{greek}{languageattribute{greek}{polutoniko}}}
4193 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}}
4194 \DeclareOption{scottishgaelic}{\bbl@try@load@lang{}{scottish}}
4195 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}}
4196 \DeclareOption{uppersorbian}{\bbl@try@load@lang{}{usorbian}}

```

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

```

4197 \ifx\bbl@opt@config\@nnil
4198   \@ifpackagewith{babel}{noconfigs}}}%
4199   {\InputIfFileExists{bblopts.cfg}%
4200     {\typeout{*****^J%
4201               * Local config file bblopts.cfg used^^J%
4202               *}}}%
4203     {}}%
4204 \else
4205   \InputIfFileExists{\bbl@opt@config.cfg}%
4206   {\typeout{*****^J%
4207             * Local config file \bbl@opt@config.cfg used^^J%
4208             *}}}%
4209   {\bbl@error{%
4210     Local config file '\bbl@opt@config.cfg' not found}%
4211     Perhaps you misspelled it.}}%
4212 \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.

```

4213 \ifx\bbl@opt@main\@nnil
4214 \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4215 \let\bbl@tempb\@empty
4216 \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4217 \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4218 \bbl@foreach\bbl@tempb{% \bbl@tempb is a reversed list
4219 \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4220 \ifodd\bbl@iniflag % = *=
4221 \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
4222 \else % n +=
4223 \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4224 \fi
4225 \fi}%
4226 \fi
4227 \else
4228 \bbl@info{Main language set with 'main='. Except if you have\\%
4229 problems, prefer the default mechanism for setting\\%
4230 the main language, ie, as the last declared.\\%
4231 Reported}
4232 \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).

```

4233 \ifx\bbl@opt@main\@nnil\else
4234 \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
4235 \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4236 \fi

```

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

```

4237 \bbl@foreach\bbl@language@opts{%
4238 \def\bbl@tempa{#1}%
4239 \ifx\bbl@tempa\bbl@opt@main\else
4240 \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4241 \bbl@ifunset{ds@#1}%
4242 {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4243 {}%
4244 \else % + * (other = ini)
4245 \DeclareOption{#1}{%
4246 \bbl@ldfinit
4247 \babelprovide[import]{#1}%
4248 \bbl@afterldf{}}%
4249 \fi
4250 \fi}
4251 \bbl@foreach\@classoptionslist{%
4252 \def\bbl@tempa{#1}%
4253 \ifx\bbl@tempa\bbl@opt@main\else
4254 \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4255 \bbl@ifunset{ds@#1}%
4256 {\IfFileExists{#1.ldf}%
4257 {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4258 {}}%
4259 {}%
4260 \else % + * (other = ini)
4261 \IfFileExists{babel-#1.tex}%
4262 {\DeclareOption{#1}{%
4263 \bbl@ldfinit
4264 \babelprovide[import]{#1}%
4265 \bbl@afterldf{}}}%
4266 {}%

```

```

4267 \fi
4268 \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):

```

4269 \def\AfterBabelLanguage#1{%
4270 \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4271 \DeclareOption*{}
4272 \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.

```

4273 \bbl@trace{Option 'main'}
4274 \ifx\bbl@opt@main\@nnil
4275 \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4276 \let\bbl@tempc\@empty
4277 \edef\bbl@templ{,\bbl@loaded,}
4278 \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4279 \bbl@for\bbl@tempb\bbl@tempa{%
4280 \edef\bbl@tempd{,\bbl@tempb,}%
4281 \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4282 \bbl@xin@{\bbl@tempd}{\bbl@templ}%
4283 \ifin\edef\bbl@tempc{\bbl@tempb}\fi}
4284 \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4285 \expandafter\bbl@tempa\bbl@loaded,\@nnil
4286 \ifx\bbl@tempb\bbl@tempc\else
4287 \bbl@warning{%
4288 Last declared language option is '\bbl@tempc',\%
4289 but the last processed one was '\bbl@tempb'.\%
4290 The main language can't be set as both a global\%
4291 and a package option. Use 'main=\bbl@tempc' as\%
4292 option. Reported}
4293 \fi
4294 \else
4295 \ifodd\bbl@iniflag % case 1,3 (main is ini)
4296 \bbl@ldfinit
4297 \let\CurrentOption\bbl@opt@main
4298 \bbl@exp{% \bbl@opt@provide = empty if *
4299 \\\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4300 \bbl@afterldf{}
4301 \DeclareOption{\bbl@opt@main}{}
4302 \else % case 0,2 (main is ldf)
4303 \ifx\bbl@loadmain\relax
4304 \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4305 \else
4306 \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4307 \fi
4308 \ExecuteOptions{\bbl@opt@main}
4309 \@namedef{ds@\bbl@opt@main}{}%
4310 \fi
4311 \DeclareOption*{}
4312 \ProcessOptions*
4313 \fi
4314 \bbl@exp{%
4315 \\\AtBeginDocument{\bbl@usehooks@lang{/}{\begindocument}{}}}%
4316 \def\AfterBabelLanguage{%
4317 \bbl@error
4318 {Too late for \string\AfterBabelLanguage}%

```

```
4319 {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.

```
4320 \ifx\bbl@main@language\undefined
4321 \bbl@info{%
4322   You haven't specified a language as a class or package\\%
4323   option. I'll load 'nil'. Reported}
4324 \bbl@load@language{nil}
4325 \fi
4326 \}
```

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`

```
4327 \*kernel
4328 \let\bbl@onlyswitch\@empty
4329 \input babel.def
4330 \let\bbl@onlyswitch\@undefined
4331 \}
```

7 Loading hyphenation patterns

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

```
4333 \MakeSureProvidesFile{is defined}
4334 \ProvidesFile{hyphen.cfg}[<date>] v<version> Babel hyphens]
4335 \xdef\bbl@format{\jobname}
4336 \def\bbl@version{<version>}
4337 \def\bbl@date{<date>}
4338 \ifx\AtBeginDocument\undefined
4339 \def\@empty{}
4340 \fi
4341 \DefineCoreSwitchingMacros
```

`\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.

```
4342 \def\process@line#1#2 #3 #4 {%
4343   \ifx=#1%
4344     \process@synonym{#2}%
4345   \else
4346     \process@language{#1#2}{#3}{#4}%
4347   \fi
4348   \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.

```

4349 \toks@{}
4350 \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.

```

4351 \def\process@synonym#1{%
4352   \ifnum\last@language=\m@ne
4353     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4354   \else
4355     \expandafter\chardef\csname l@#1\endcsname\last@language
4356     \wlog{\string\l@#1=\string\language\the\last@language}%
4357     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4358       \csname\language\hyphenmins\endcsname
4359     \let\bbl@elt\relax
4360     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4361   \fi}

```

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

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

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

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

Pattern files may contain assignments to \lefthyphenmin and \righthyphenmin. T_EX 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.

```

4362 \def\process@language#1#2#3{%
4363   \expandafter\addlanguage\csname l@#1\endcsname
4364   \expandafter\language\csname l@#1\endcsname
4365   \edef\language{#1}%
4366   \bbl@hook@everylanguage{#1}%
4367   % > luatex
4368   \bbl@get@enc#1: :@@@
4369   \begingroup
4370     \lefthyphenmin\m@ne
4371     \bbl@hook@loadpatterns{#2}%
4372     % > luatex
4373     \ifnum\lefthyphenmin=\m@ne
4374     \else
4375       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4376         \the\lefthyphenmin\the\righthyphenmin}%
4377     \fi

```

```

4378 \endgroup
4379 \def\bbl@tempa{#3}%
4380 \ifx\bbl@tempa\@empty\else
4381   \bbl@hook@loadexceptions{#3}%
4382   % > luatex
4383 \fi
4384 \let\bbl@elt\relax
4385 \edef\bbl@languages{%
4386   \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4387 \ifnum\the\language=\z@
4388   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4389     \set@hyphenmins\tw@\thr@@\relax
4390   \else
4391     \expandafter\expandafter\expandafter\set@hyphenmins
4392     \csname #1hyphenmins\endcsname
4393   \fi
4394   \the\toks@
4395   \toks@{}%
4396 \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.

```

4397 \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.

```

4398 \def\bbl@hook@everylanguage#1{}
4399 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4400 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4401 \def\bbl@hook@loadkernel#1{%
4402   \def\addlanguage{\csname newlanguage\endcsname}%
4403   \def\adddialect##1##2{%
4404     \global\chardef##1##2\relax
4405     \wlog{\string##1 = a dialect from \string\language##2}}%
4406   \def\iflanguage##1{%
4407     \expandafter\ifx\csname l@##1\endcsname\relax
4408       \nolannerr{##1}%
4409     \else
4410       \ifnum\csname l@##1\endcsname=\language
4411         \expandafter\expandafter\expandafter\@firstoftwo
4412       \else
4413         \expandafter\expandafter\expandafter\@secondoftwo
4414       \fi
4415     \fi}%
4416   \def\providehyphenmins##1##2{%
4417     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4418       \@namedef{##1hyphenmins}{##2}%
4419     \fi}%
4420   \def\set@hyphenmins##1##2{%
4421     \lefthyphenmin##1\relax
4422     \righthyphenmin##2\relax}%
4423   \def\selectlanguage{%
4424     \errhelp{Selecting a language requires a package supporting it}%
4425     \errmessage{Not loaded}}%
4426   \let\foreignlanguage\selectlanguage
4427   \let\otherlanguage\selectlanguage
4428   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4429   \def\bbl@usehooks##1##2{% TODO. Temporary!!
4430     \def\setlocale{%
4431       \errhelp{Find an armchair, sit down and wait}%
4432       \errmessage{Not yet available}}%
4433     \let\uselocale\setlocale

```



```

4434 \let\locale\setlocale
4435 \let\selectlocale\setlocale
4436 \let\localename\setlocale
4437 \let\textlocale\setlocale
4438 \let\textlanguage\setlocale
4439 \let\languagegettext\setlocale}
4440 \begingroup
4441 \def\AddBabelHook#1#2{%
4442   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4443     \def\next{\toks1}%
4444   \else
4445     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4446   \fi
4447   \next}
4448 \ifx\directlua@undefined
4449   \ifx\XeTeXinputencoding@undefined\else
4450     \input xebabel.def
4451   \fi
4452 \else
4453   \input luababel.def
4454 \fi
4455 \openin1 = babel-\bbl@format.cfg
4456 \ifeof1
4457 \else
4458   \input babel-\bbl@format.cfg\relax
4459 \fi
4460 \closein1
4461 \endgroup
4462 \bbl@hook@loadkernel{switch.def}

```

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

```

4463 \openin1 = language.dat

```

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

```

4464 \def\language{english}%
4465 \ifeof1
4466   \message{I couldn't find the file language.dat,\space
4467     I will try the file hyphen.tex}
4468   \input hyphen.tex\relax
4469   \chardef\l@english\z@
4470 \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`.

```

4471 \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.

```

4472 \loop
4473   \endlinechar\m@ne
4474   \read1 to \bbl@line
4475   \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.

```

4476   \if T\ifeof1F\fi T\relax
4477   \ifx\bbl@line@empty\else
4478     \edef\bbl@line{\bbl@line\space\space\space}%
4479     \expandafter\process@line\bbl@line\relax

```

```

4480     \fi
4481 \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.

```

4482 \begingroup
4483   \def\bbl@elt#1#2#3#4{%
4484     \global\language=#2\relax
4485     \gdef\language#1{%
4486       \def\bbl@elt##1##2##3##4{}}%
4487   \bbl@languages
4488 \endgroup
4489 \fi
4490 \closeinl

```

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

```

4491 \if/\the\toks@/\else
4492   \errhelp{language.dat loads no language, only synonyms}
4493   \errmessage{Orphan language synonym}
4494 \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.

```

4495 \let\bbl@line@undefined
4496 \let\process@line@undefined
4497 \let\process@synonym@undefined
4498 \let\process@language@undefined
4499 \let\bbl@get@enc@undefined
4500 \let\bbl@hyph@enc@undefined
4501 \let\bbl@tempa@undefined
4502 \let\bbl@hook@loadkernel@undefined
4503 \let\bbl@hook@everylanguage@undefined
4504 \let\bbl@hook@loadpatterns@undefined
4505 \let\bbl@hook@loadexceptions@undefined
4506 \patterns

```

Here the code for `iniTeX` ends.

8 Font handling with fontspec

Add the `bid` handler just before `luaotfload`, 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 `bid` [misplaced].

```

4507 <<More package options>> ≡
4508 \chardef\bbl@bidimode\z@
4509 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4510 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4511 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4512 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4513 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4514 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4515 <</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.

```

4516 <<Font selection>> ≡
4517 \bbl@trace{Font handling with fontspec}
4518 \ifx\ExplSyntaxOn\@undefined\else
4519   \def\bbl@fs@warn@nx#1#2{% \bbl@tempfs is the original macro

```

```

4520 \in@{, #1,}{, no-script, language-not-exist,}%
4521 \ifin@ \else \bbl@tempfs@nx{#1}{#2}\fi}
4522 \def\bbl@fs@warn@nx#1#2#3{%
4523 \in@{, #1,}{, no-script, language-not-exist,}%
4524 \ifin@ \else \bbl@tempfs@nx{#1}{#2}{#3}\fi}
4525 \def\bbl@loadfontspec{%
4526 \let\bbl@loadfontspec\relax
4527 \ifx\fontspec\undefined
4528 \usepackage{fontspec}%
4529 \fi}%
4530 \fi
4531 \@onlypreamble\babelfont
4532 \newcommand\babelfont[2][{}% 1=langs/scripts 2=fam
4533 \bbl@foreach{#1}{%
4534 \expandafter\ifx\csname date##1\endcsname\relax
4535 \IfFileExists{babel-##1.tex}%
4536 {\babelprovide{##1}}%
4537 {}%
4538 \fi}%
4539 \edef\bbl@tempa{#1}%
4540 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4541 \bbl@loadfontspec
4542 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4543 \bbl@bblfont}
4544 \newcommand\bbl@bblfont[2][{}% 1=features 2=fontname, @font=rm|sf|tt
4545 \bbl@ifunset{\bbl@tempb family}%
4546 {\bbl@providefam{\bbl@tempb}}%
4547 {}%
4548 % For the default font, just in case:
4549 \bbl@ifunset{\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4550 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4551 {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save \bbl@rmdflt@
4552 \bbl@exp{%
4553 \let<\bbl@\bbl@tempb dflt@\languagename>\<\bbl@\bbl@tempb dflt@>%
4554 \\\bbl@font@set<\bbl@\bbl@tempb dflt@\languagename>%
4555 <\bbl@tempb default>\<\bbl@tempb family>}}%
4556 {\bbl@foreach\bbl@tempa{% ie \bbl@rmdflt@lang / *scrt
4557 \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:

```

4558 \def\bbl@providefam#1{%
4559 \bbl@exp{%
4560 \\\newcommand<#1default>{}% Just define it
4561 \\\bbl@add@list\\bbl@font@fams{#1}%
4562 \\\DeclareRobustCommand<#1family>{%
4563 \\\not@math@alphabet<#1family>\relax
4564 % \\\prepare@family@series@update{#1}<#1default>% TODO. Fails
4565 \\\fontfamily<#1default>%
4566 <\ifx>\\UseHooks\\@undefined\<else>\\UseHook{#1family}\<fi>%
4567 \\\selectfont}%
4568 \\\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.

```

4569 \def\bbl@nostdfont#1{%
4570 \bbl@ifunset{\bbl@WFF@f@family}%
4571 {\bbl@csarg\gdef{WFF@f@family}}{}% Flag, to avoid dupl warns
4572 \bbl@infowarn{The current font is not a babel standard family:\\%
4573 #1%
4574 \fontname\font\\%
4575 There is nothing intrinsically wrong with this warning, and\\%
4576 you can ignore it altogether if you do not need these\\%
4577 families. But if they are used in the document, you should be\\%
4578 aware 'babel' will not set Script and Language for them, so\\%

```

```

4579     you may consider defining a new family with \string\babelfont.\\%
4580     See the manual for further details about \string\babelfont.\\%
4581     Reported}}
4582   {}}%
4583 \gdef\bbbl@switchfont{%
4584   \bbbl@ifunset{bbbl@lsys@\languageName}{\bbbl@provide@lsys{\languageName}}{}%
4585   \bbbl@exp{% eg Arabic -> arabic
4586     \lowercase{\edef\\bbbl@tempa{\bbbl@cl{sname}}}}}%
4587   \bbbl@foreach\bbbl@font@fams{%
4588     \bbbl@ifunset{bbbl@##ldflt@\languageName}%      (1) language?
4589     {\bbbl@ifunset{bbbl@##ldflt@*\bbbl@tempa}%      (2) from script?
4590     {\bbbl@ifunset{bbbl@##ldflt@}%                  2=F - (3) from generic?
4591     {}%                                              123=F - nothing!
4592     {\bbbl@exp{%                                    3=T - from generic
4593       \global\let<\bbbl@##ldflt@\languageName>%
4594       \<\bbbl@##ldflt@>}}}%
4595     {\bbbl@exp{%                                    2=T - from script
4596       \global\let<\bbbl@##ldflt@\languageName>%
4597       \<\bbbl@##ldflt@*\bbbl@tempa>}}}%
4598     {}}%                                          1=T - language, already defined
4599   \def\bbbl@tempa{\bbbl@nostdfont{}}% TODO. Don't use \bbbl@tempa
4600   \bbbl@foreach\bbbl@font@fams{% don't gather with prev for
4601     \bbbl@ifunset{bbbl@##ldflt@\languageName}%
4602     {\bbbl@cs{famrst@##1}%
4603     \global\bbbl@csarg\let{famrst@##1}\relax}%
4604     {\bbbl@exp{% order is relevant. TODO: but sometimes wrong!
4605     \\bbbl@add\\originalTeX{%
4606     \\bbbl@font@rst{\bbbl@cl{##ldflt}}}%
4607     \<##ldflt>\<##lfamily>{##1}}}%
4608     \\bbbl@font@set<\bbbl@##ldflt@\languageName>% the main part!
4609     \<##ldflt>\<##lfamily>}}}%
4610   \bbbl@ifrestoring{ }\bbbl@tempa}%

```

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

```

4611 \ifx\f@family\undefined\else % if latex
4612 \ifcase\bbbl@engine % if pdftex
4613 \let\bbbl@ckeckstdfonts\relax
4614 \else
4615 \def\bbbl@ckeckstdfonts{%
4616   \begingroup
4617   \global\let\bbbl@ckeckstdfonts\relax
4618   \let\bbbl@tempa\empty
4619   \bbbl@foreach\bbbl@font@fams{%
4620     \bbbl@ifunset{bbbl@##ldflt@}%
4621     {\@nameuse{##lfamily}%
4622     \bbbl@csarg\gdef{WFF@f@family}}}% Flag
4623     \bbbl@exp{\\bbbl@add\\bbbl@tempa{* \<##lfamily>= \f@family\\}%
4624     \space\space\fontname\font\\}%
4625     \bbbl@csarg\xdef{##ldflt@}{\f@family}%
4626     \expandafter\xdef\csname ##ldflt\endcsname{\f@family}%
4627     {}}%
4628   \ifx\bbbl@tempa\empty\else
4629   \bbbl@infowarn{The following font families will use the default\\%
4630     settings for all or some languages:\\%
4631     \bbbl@tempa
4632     There is nothing intrinsically wrong with it, but\\%
4633     'babel' will no set Script and Language, which could\\%
4634     be relevant in some languages. If your document uses\\%
4635     these families, consider redefining them with \string\babelfont.\\%
4636     Reported}%
4637   \fi
4638   \endgroup}

```

```
4639 \fi
4640 \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.

```
4641 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4642 \bbl@xin@{<>}{#1}%
4643 \ifin@
4644 \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4645 \fi
4646 \bbl@exp{% 'Unprotected' macros return prev values
4647 \def\\#2{#1}% eg, \rmdefault{\bbl@rmdflt@lang}
4648 \\bbl@ifsamestring{#2}{\f@family}%
4649 {\#3%
4650 \\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}{}%
4651 \let\\bbl@tempa\relax}%
4652 {}}}
4653 % TODO - next should be global?, but even local does its job. I'm
4654 % still not sure -- must investigate:
4655 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4656 \let\bbl@tempe\bbl@mapselect
4657 \edef\bbl@tempb{\bbl@stripslash#4/}% Catcodes hack (better pass it).
4658 \bbl@exp{\\bbl@replace\\bbl@tempb{\bbl@stripslash\family/}{}}%
4659 \let\bbl@mapselect\relax
4660 \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4661 \let#4\@empty % Make sure \renewfontfamily is valid
4662 \bbl@exp{%
4663 \let\\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
4664 <\keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4665 {\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4666 <\keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4667 {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4668 \let\\bbl@tempfs@nx<__fontspec_warning:nx>%
4669 \let<__fontspec_warning:nx>\\bbl@fs@warn@nx
4670 \let\\bbl@tempfs@nxx<__fontspec_warning:nxx>%
4671 \let<__fontspec_warning:nxx>\\bbl@fs@warn@nxx
4672 \\renewfontfamily\\#4%
4673 [\bbl@cl{lsys},%
4674 \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
4675 #2}}{#3}% ie \bbl@exp{..}{#3}
4676 \bbl@exp{%
4677 \let<__fontspec_warning:nx>\\bbl@tempfs@nx
4678 \let<__fontspec_warning:nxx>\\bbl@tempfs@nxx}%
4679 \beginngroup
4680 #4%
4681 \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4682 \endgroup
4683 \let#4\bbl@temp@fam
4684 \bbl@exp{\let<\bbl@stripslash#4\space>\bbl@temp@pfam
4685 \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.

```
4686 \def\bbl@font@rst#1#2#3#4{%
4687 \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`.

```
4688 \def\bbl@font@fams{rm,sf,tt}
4689 <</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.

```
4690 <<{*Footnote changes}>> ≡
4691 \bbl@trace{Bidi footnotes}
4692 \ifnum\bbl@bidimode>\z@ % Any bidi=
4693 \def\bbl@footnote#1#2#3{%
4694   \ifnextchar[%
4695     {\bbl@footnote@o{#1}{#2}{#3}}%
4696     {\bbl@footnote@x{#1}{#2}{#3}}}
4697 \long\def\bbl@footnote@x#1#2#3#4{%
4698   \bgroup
4699   \select@language@x{\bbl@main@language}%
4700   \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4701   \egroup}
4702 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4703   \bgroup
4704   \select@language@x{\bbl@main@language}%
4705   \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4706   \egroup}
4707 \def\bbl@footnotetext#1#2#3{%
4708   \ifnextchar[%
4709     {\bbl@footnotetext@o{#1}{#2}{#3}}%
4710     {\bbl@footnotetext@x{#1}{#2}{#3}}}
4711 \long\def\bbl@footnotetext@x#1#2#3#4{%
4712   \bgroup
4713   \select@language@x{\bbl@main@language}%
4714   \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4715   \egroup}
4716 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4717   \bgroup
4718   \select@language@x{\bbl@main@language}%
4719   \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4720   \egroup}
4721 \def\BabelFootnote#1#2#3#4{%
4722   \ifx\bbl@fn@footnote\undefined
4723     \let\bbl@fn@footnote\footnote
4724   \fi
4725   \ifx\bbl@fn@footnotetext\undefined
4726     \let\bbl@fn@footnotetext\footnotetext
4727   \fi
4728   \bbl@ifblank{#2}%
4729     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4730     \@namedef{\bbl@stripslash#1text}%
4731     {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4732     {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4733     \@namedef{\bbl@stripslash#1text}%
4734     {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4735 \fi
4736 <</Footnote changes>>
```

Now, the code.

```
4737 <{*xetex}>
4738 \def\BabelStringsDefault{unicode}
4739 \let\xebbl@stop\relax
4740 \AddBabelHook{xetex}{encodedcommands}{%
4741   \def\bbl@tempa{#1}%
4742   \ifx\bbl@tempa\empty
4743     \XeTeXinputencoding"bytes"%
4744   \else
```

```

4745 \XeTeXinputencoding"#1"%
4746 \fi
4747 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4748 \AddBabelHook{xetex}{stopcommands}{%
4749 \xebbl@stop
4750 \let\xebbl@stop\relax}
4751 \def\bbl@intraspace#1 #2 #3\@@{%
4752 \bbl@csarg\gdef{xeisp@\language}%
4753 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4754 \def\bbl@intrapenalty#1\@@{%
4755 \bbl@csarg\gdef{xeipn@\language}%
4756 {\XeTeXlinebreakpenalty #1\relax}}
4757 \def\bbl@provide@intraspace{%
4758 \bbl@xin@{/s}{/\bbl@c{l}{lnbrk}}}%
4759 \ifin@ \else \bbl@xin@{/c}{/\bbl@c{l}{lnbrk}} \fi
4760 \ifin@
4761 \bbl@ifunset{bbl@intsp@\language}{}%
4762 {\expandafter\ifx\cname bbl@intsp@\language\endcsname\empty\else
4763 \ifx\bbl@KVP@intraspace\@nnil
4764 \bbl@exp{%
4765 \\\bbl@intraspace\bbl@c{l}{intsp}}\@@}%
4766 \fi
4767 \ifx\bbl@KVP@intrapenalty\@nnil
4768 \bbl@intrapenalty0\@@
4769 \fi
4770 \fi
4771 \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4772 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4773 \fi
4774 \ifx\bbl@KVP@intrapenalty\@nnil\else
4775 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4776 \fi
4777 \bbl@exp{%
4778 % TODO. Execute only once (but redundant):
4779 \\\bbl@add\<extras\language>{%
4780 \XeTeXlinebreaklocale "\bbl@c{l}{tbcpr}"%
4781 \<bbl@xeisp@\language>%
4782 \<bbl@xeipn@\language>}%
4783 \\\bbl@together\<extras\language>%
4784 \\\bbl@add\<noextras\language>{%
4785 \XeTeXlinebreaklocale ""}%
4786 \\\bbl@together\<noextras\language>}%
4787 \ifx\bbl@ispace\@undefined
4788 \gdef\bbl@ispace{\bbl@c{l}{xeisp}}%
4789 \ifx\AtBeginDocument\@notprerr
4790 \expandafter\@secondoftwo % to execute right now
4791 \fi
4792 \AtBeginDocument{\bbl@patchfont{\bbl@ispace}}%
4793 \fi}%
4794 \fi}
4795 \ifx\DisableBabelHook\@undefined\endinput\fi
4796 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4797 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@c{ckcstdfont}}
4798 \DisableBabelHook{babel-fontspec}
4799 <<Font selection>>
4800 \def\bbl@provide@extra#1{}
4801 </xetex>

```

9.2 Layout

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

`\bbl@startskip` and `\bbl@endskip` are available to package authors. Thanks to the \TeX expansion

mechanism the following constructs are valid: \adim\bbl@startskip,
\advance\bbl@startskip\adim, \bbl@startskip\adim.
Consider txtbabel as a shorthand for *tex-xet babel*, which is the bidi model in both pdftex and xetex.

```

4802 (*xetex | texxet)
4803 \providecommand\bbl@provide@intraspace{}
4804 \bbl@trace{Redefinitions for bidi layout}
4805 \def\bbl@sspre@caption{%
4806   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4807 \ifx\bbl@opt@layout\@nnil\else % if layout=..
4808 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4809 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4810 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4811 \def\@hangfrom#1{%
4812   \setbox\@tempboxa\hbox{#1}%
4813   \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4814   \noindent\box\@tempboxa}
4815 \def\raggedright{%
4816   \let\\\@centercr
4817   \bbl@startskip\z@skip
4818   \@rightskip\@flushglue
4819   \bbl@endskip\@rightskip
4820   \parindent\z@
4821   \parfillskip\bbl@startskip}
4822 \def\raggedleft{%
4823   \let\\\@centercr
4824   \bbl@startskip\@flushglue
4825   \bbl@endskip\z@skip
4826   \parindent\z@
4827   \parfillskip\bbl@endskip}
4828 \fi
4829 \IfBabelLayout{lists}
4830 {\bbl@sreplace\list
4831   {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4832   \def\bbl@listleftmargin{%
4833     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4834   \ifcase\bbl@engine
4835     \def\labelenumii{}\theenumii}% pdftex doesn't reverse ()
4836     \def\p@enumiii{\p@enumii}\theenumii}%
4837   \fi
4838   \bbl@sreplace\@verbatim
4839     {\leftskip\@totalleftmargin}%
4840     {\bbl@startskip\textwidth
4841       \advance\bbl@startskip-\linewidth}%
4842   \bbl@sreplace\@verbatim
4843     {\rightskip\z@skip}%
4844     {\bbl@endskip\z@skip}}%
4845 {}
4846 \IfBabelLayout{contents}
4847 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4848   \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4849 {}
4850 \IfBabelLayout{columns}
4851 {\bbl@sreplace\@outputdblcol{\hbext@\textwidth}{\bbl@outpuhbox}%
4852   \def\bbl@outpuhbox#1{%
4853     \hbext@\textwidth%
4854     \hskip\columnwidth
4855     \hfil
4856     {\normalcolor\vrule \@width\columnseprule}%
4857     \hfil
4858     \hbext@\columnwidth{\box\@leftcolumn \hss}%
4859     \hskip-\textwidth
4860     \hbext@\columnwidth{\box\@outputbox \hss}%
4861     \hskip\columnsep

```



```

4862 \hskip\columnwidth}}}%
4863 {}
4864 <<Footnote changes>>
4865 \IfBabelLayout{footnotes}%
4866 {\BabelFootnote\footnote\languagename{}}}%
4867 \BabelFootnote\localfootnote\languagename{}}}%
4868 \BabelFootnote\mainfootnote{}}{}
4869 {}

```

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.

```

4870 \IfBabelLayout{counters*}%
4871 {\bbl@add\bbl@opt@layout{.counters.}%
4872 \AddToHook{shipout/before}{%
4873 \let\bbl@tempa\babelsublr
4874 \let\babelsublr\@firstofone
4875 \let\bbl@save@thepage\thepage
4876 \protected@edef\thepage{\thepage}%
4877 \let\babelsublr\bbl@tempa}%
4878 \AddToHook{shipout/after}{%
4879 \let\thepage\bbl@save@thepage}}{}
4880 \IfBabelLayout{counters}%
4881 {\let\bbl@latinarabic=\@arabic
4882 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
4883 \let\bbl@asciroman=\@roman
4884 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4885 \let\bbl@asciiRoman=\@Roman
4886 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
4887 \fi % end if layout
4888 </xetex | texpet>

```

9.3 8-bit TeX

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

```

4889 <*texpet>
4890 \def\bbl@provide@extra#1{%
4891 % == auto-select encoding ==
4892 \ifx\bbl@encoding@select@off\@empty\else
4893 \bbl@ifunset{\bbl@encoding@#1}%
4894 {\def\@elt##1{,##1,}%
4895 \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
4896 \count@\z@
4897 \bbl@foreach\bbl@tempe{%
4898 \def\bbl@tempd{##1}% Save last declared
4899 \advance\count@\@ne}%
4900 \ifnum\count@>\@ne
4901 \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
4902 \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
4903 \bbl@replace\bbl@tempa{ },}%
4904 \global\bbl@csarg\let{encoding@#1}\@empty
4905 \bbl@xin@{\bbl@tempd,}{,\bbl@tempa,}%
4906 \ifin@\else % if main encoding included in ini, do nothing
4907 \let\bbl@tempb\relax
4908 \bbl@foreach\bbl@tempa{%
4909 \ifx\bbl@tempb\relax
4910 \bbl@xin@{,##1,}{,\bbl@tempe,}%
4911 \ifin@\def\bbl@tempb{##1}\fi
4912 \fi}%
4913 \ifx\bbl@tempb\relax\else
4914 \bbl@exp{%
4915 \global\<\bbl@add>\<\bbl@preextras@#1>\<\bbl@encoding@#1>%
4916 \gdef\<\bbl@encoding@#1>{%
4917 \\\babel@save\\f@encoding

```

```

4918          \\bbl@add\\originalTeX{\\selectfont}%
4919          \\fontencoding{\\bbl@tempb}%
4920          \\selectfont}}}%
4921      \fi
4922      \fi
4923      \fi}%
4924      {}%
4925  \fi}
4926 </texxet>

```

9.4 LuaTeX

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

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

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

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

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

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

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

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

```

4927 <*\luatex>
4928 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4929 \bbl@trace{Read language.dat}
4930 \ifx\bbl@readstream\undefined
4931   \csname newread\endcsname\bbl@readstream
4932 \fi
4933 \beginngroup
4934   \toks@{}
4935   \count@ \z@ % 0=start, 1=0th, 2=normal
4936   \def\bbl@process@line#1#2 #3 #4 {%
4937     \ifx=#1%
4938       \bbl@process@synonym{#2}%
4939     \else
4940       \bbl@process@language{#1#2}{#3}{#4}%
4941     \fi
4942     \ignorespaces}
4943   \def\bbl@manylang{%
4944     \ifnum\bbl@last>\@ne
4945       \bbl@info{Non-standard hyphenation setup}%

```

```

4946 \fi
4947 \let\bbl@manylang\relax}
4948 \def\bbl@process@language#1#2#3{%
4949 \ifcase\count@
4950 \ifundefined{zth@#1}{\count@tw@}{\count@ne}%
4951 \or
4952 \count@tw@
4953 \fi
4954 \ifnum\count@=\tw@
4955 \expandafter\addlanguage\csname l@#1\endcsname
4956 \language\allocationnumber
4957 \chardef\bbl@last\allocationnumber
4958 \bbl@manylang
4959 \let\bbl@elt\relax
4960 \xdef\bbl@languages{%
4961 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4962 \fi
4963 \the\toks@
4964 \toks@{}}
4965 \def\bbl@process@synonym@aux#1#2{%
4966 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4967 \let\bbl@elt\relax
4968 \xdef\bbl@languages{%
4969 \bbl@languages\bbl@elt{#1}{#2}{}}}%
4970 \def\bbl@process@synonym#1{%
4971 \ifcase\count@
4972 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4973 \or
4974 \ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}}%
4975 \else
4976 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4977 \fi}
4978 \ifx\bbl@languages\undefined % Just a (sensible?) guess
4979 \chardef\l@english\z@
4980 \chardef\l@USenglish\z@
4981 \chardef\bbl@last\z@
4982 \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
4983 \gdef\bbl@languages{%
4984 \bbl@elt{english}{0}{hyphen.tex}}%
4985 \bbl@elt{USenglish}{0}{}}
4986 \else
4987 \global\let\bbl@languages@format\bbl@languages
4988 \def\bbl@elt#1#2#3#4{% Remove all except language 0
4989 \ifnum#2>\z@\else
4990 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4991 \fi}%
4992 \xdef\bbl@languages{\bbl@languages}%
4993 \fi
4994 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}} % Define flags
4995 \bbl@languages
4996 \openin\bbl@readstream=language.dat
4997 \ifeof\bbl@readstream
4998 \bbl@warning{I couldn't find language.dat. No additional\\%
4999 patterns loaded. Reported}%
5000 \else
5001 \loop
5002 \endlinechar\m@ne
5003 \read\bbl@readstream to \bbl@line
5004 \endlinechar`^^M
5005 \if T\ifeof\bbl@readstream F\fi T\relax
5006 \ifx\bbl@line\empty\else
5007 \edef\bbl@line{\bbl@line\space\space\space}%
5008 \expandafter\bbl@process@line\bbl@line\relax

```

```

5009     \fi
5010   \repeat
5011 \fi
5012 \closein\bbL@readstream
5013 \endgroup
5014 \bbL@trace{Macros for reading patterns files}
5015 \def\bbL@get@enc#1:#2:#3\@@{\def\bbL@hyph@enc{#2}}
5016 \ifx\babelcatcodetablenum\undefined
5017   \ifx\newcatcodetable\undefined
5018     \def\babelcatcodetablenum{5211}
5019     \def\bbL@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5020   \else
5021     \newcatcodetable\babelcatcodetablenum
5022     \newcatcodetable\bbL@pattcodes
5023   \fi
5024 \else
5025   \def\bbL@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5026 \fi
5027 \def\bbL@luapatterns#1#2{%
5028   \bbL@get@enc#1:.\@@@
5029   \setbox\z@\hbox\bgroup
5030     \begingroup
5031       \savecatcodetable\babelcatcodetablenum\relax
5032       \initcatcodetable\bbL@pattcodes\relax
5033       \catcodetable\bbL@pattcodes\relax
5034       \catcode\#=6 \catcode\$=3 \catcode\&=4 \catcode\^=7
5035       \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\~=13
5036       \catcode\@=11 \catcode\^^I=10 \catcode\^^J=12
5037       \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
5038       \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
5039       \catcode\`=12 \catcode\'=12 \catcode\"=12
5040       \input #1\relax
5041       \catcodetable\babelcatcodetablenum\relax
5042     \endgroup
5043   \def\bbL@tempa{#2}%
5044   \ifx\bbL@tempa\@empty\else
5045     \input #2\relax
5046   \fi
5047 \egroup}%
5048 \def\bbL@patterns@lua#1{%
5049   \language=\expandafter\ifx\csname l@#1:f@encoding\endcsname\relax
5050     \csname l@#1\endcsname
5051     \edef\bbL@tempa{#1}%
5052   \else
5053     \csname l@#1:f@encoding\endcsname
5054     \edef\bbL@tempa{#1:f@encoding}%
5055   \fi\relax
5056   \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5057   \@ifundefined{bbL@hyphendata@the\language}%
5058     {\def\bbL@elt##1##2##3##4{%
5059       \ifnum##2=\csname l@bbL@tempa\endcsname % #2=spanish, dutch:OT1...
5060       \def\bbL@tempb{##3}%
5061       \ifx\bbL@tempb\@empty\else % if not a synonymous
5062         \def\bbL@tempc{{##3}{##4}}}%
5063       \fi
5064       \bbL@csarg\xdef{hyphendata@##2}{\bbL@tempc}%
5065     \fi}%
5066   \bbL@languages
5067   \@ifundefined{bbL@hyphendata@the\language}%
5068     {\bbL@info{No hyphenation patterns were set for\\%
5069       language '\bbL@tempa'. Reported}}%
5070     {\expandafter\expandafter\expandafter\bbL@luapatterns
5071       \csname bbL@hyphendata@the\language\endcsname}}}}

```

```

5072 \endinput\fi
5073 % Here ends \ifx\AddBabelHook\@undefined
5074 % A few lines are only read by hyphen.cfg
5075 \ifx\DisableBabelHook\@undefined
5076 \AddBabelHook{luatex}{everylanguage}{%
5077   \def\process@language##1##2##3{%
5078     \def\process@line####1####2 ####3 ####4 {}}}
5079 \AddBabelHook{luatex}{loadpatterns}{%
5080   \input #1\relax
5081   \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5082     {#{1}}{}}
5083 \AddBabelHook{luatex}{loadexceptions}{%
5084   \input #1\relax
5085   \def\bbl@tempb##1##2{#{1}}{#{1}}%
5086   \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5087     {\expandafter\expandafter\expandafter\bbl@tempb
5088       \csname bbl@hyphendata@the\language\endcsname}}
5089 \endinput\fi
5090 % Here stops reading code for hyphen.cfg
5091 % The following is read the 2nd time it's loaded
5092 \begingroup % TODO - to a lua file
5093 \catcode`\%=12
5094 \catcode`\'=12
5095 \catcode`\%=12
5096 \catcode`\:=12
5097 \directlua{
5098   Babel = Babel or {}
5099   function Babel.bytes(line)
5100     return line:gsub(".",
5101       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5102   end
5103   function Babel.begin_process_input()
5104     if luatexbase and luatexbase.add_to_callback then
5105       luatexbase.add_to_callback('process_input_buffer',
5106         Babel.bytes, 'Babel.bytes')
5107     else
5108       Babel.callback = callback.find('process_input_buffer')
5109       callback.register('process_input_buffer', Babel.bytes)
5110     end
5111   end
5112   function Babel.end_process_input ()
5113     if luatexbase and luatexbase.remove_from_callback then
5114       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5115     else
5116       callback.register('process_input_buffer', Babel.callback)
5117     end
5118   end
5119   function Babel.addpatterns(pp, lg)
5120     local lg = lang.new(lg)
5121     local pats = lang.patterns(lg) or ''
5122     lang.clear_patterns(lg)
5123     for p in pp:gmatch('[^%s]+') do
5124       ss = ''
5125       for i in string.utfcharacters(p:gsub('%d', '')) do
5126         ss = ss .. '%d?' .. i
5127       end
5128       ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
5129       ss = ss:gsub('%.%d%?$', '%%.')
5130       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5131       if n == 0 then
5132         tex.sprint(
5133           [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5134           .. p .. [[]])

```

```

5135     pats = pats .. ' ' .. p
5136   else
5137     tex.sprint(
5138       [[\string\csname\space bbl@info\endcsname{Renew pattern: }]
5139       .. p .. [{}]])
5140   end
5141 end
5142 lang.patterns(lg, pats)
5143 end
5144 Babel.characters = Babel.characters or {}
5145 Babel.ranges = Babel.ranges or {}
5146 function Babel.hlist_has_bidi(head)
5147   local has_bidi = false
5148   local ranges = Babel.ranges
5149   for item in node.traverse(head) do
5150     if item.id == node.id'glyph' then
5151       local itemchar = item.char
5152       local chardata = Babel.characters[itemchar]
5153       local dir = chardata and chardata.d or nil
5154       if not dir then
5155         for nn, et in ipairs(ranges) do
5156           if itemchar < et[1] then
5157             break
5158           elseif itemchar <= et[2] then
5159             dir = et[3]
5160             break
5161           end
5162         end
5163       end
5164       if dir and (dir == 'al' or dir == 'r') then
5165         has_bidi = true
5166       end
5167     end
5168   end
5169   return has_bidi
5170 end
5171 function Babel.set_chranges_b (script, chrng)
5172   if chrng == '' then return end
5173   texio.write('Replacing ' .. script .. ' script ranges')
5174   Babel.script_blocks[script] = {}
5175   for s, e in string.gmatch(chrng..' ', '(.-%.(-)%s') do
5176     table.insert(
5177       Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5178   end
5179 end
5180 function Babel.discard_sublr(str)
5181   if str:find( [[\string\indexentry]] ) and
5182     str:find( [[\string\babelsublr]] ) then
5183     str = str:gsub( [[\string\babelsublr%s*(%b{})]],
5184       function(m) return m:sub(2,-2) end )
5185   end
5186   return str
5187 end
5188 }
5189 \endgroup
5190 \ifx\newattribute\undefined\else
5191   \newattribute\bbl@attr@locale
5192   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5193   \AddBabelHook{luatex}{beforeextras}{%
5194     \setattribute\bbl@attr@locale\localeid}
5195 \fi
5196 \def\BabelStringsDefault{unicode}
5197 \let\luabbl@stop\relax

```

```

5198 \AddBabelHook{luatex}{encodedcommands}{%
5199   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5200   \ifx\bbl@tempa\bbl@tempb\else
5201     \directlua{Babel.begin_process_input()}%
5202     \def\luabbl@stop{%
5203       \directlua{Babel.end_process_input()}}%
5204   \fi}%
5205 \AddBabelHook{luatex}{stopcommands}{%
5206   \luabbl@stop
5207   \let\luabbl@stop\relax}
5208 \AddBabelHook{luatex}{patterns}{%
5209   \@ifundefined{bbl@hyphendata@the\language}%
5210     {\def\bbl@elt##1##2##3##4{%
5211       \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
5212       \def\bbl@tempb{##3}%
5213       \ifx\bbl@tempb@empty\else % if not a synonymous
5214         \def\bbl@tempc{{##3}{##4}}%
5215         \fi
5216         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5217       \fi}%
5218     \bbl@languages
5219     \@ifundefined{bbl@hyphendata@the\language}%
5220       {\bbl@info{No hyphenation patterns were set for\%
5221         language '#2'. Reported}}%
5222       {\expandafter\expandafter\expandafter\bbl@luapatterns
5223         \csname bbl@hyphendata@the\language\endcsname}}}%
5224   \@ifundefined{bbl@patterns@}{}%
5225   \begingroup
5226     \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5227     \ifin@else
5228       \ifx\bbl@patterns@empty\else
5229         \directlua{ Babel.addpatterns(
5230           [[\bbl@patterns@]], \number\language) }%
5231       \fi
5232       \@ifundefined{bbl@patterns@#1}%
5233       \@empty
5234       {\directlua{ Babel.addpatterns(
5235         [[\space\csname bbl@patterns@#1\endcsname]],
5236         \number\language) }}%
5237       \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5238     \fi
5239   \endgroup}%
5240 \bbl@exp{%
5241   \bbl@ifunset{bbl@prehc@languagename}{}%
5242   {\bbl@ifblank{\bbl@cs{prehc@languagename}}{}}%
5243   {\prehyphenchar=\bbl@c{l{prehc}}\relax}}}%

```

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

```

5244 \@onlypreamble\babelpatterns
5245 \AtEndOfPackage{%
5246   \newcommand\babelpatterns[2][\@empty]{%
5247     \ifx\bbl@patterns@relax
5248       \let\bbl@patterns@empty
5249     \fi
5250     \ifx\bbl@pttnlistempty\else
5251       \bbl@warning{%
5252         You must not intermingle \string\selectlanguage\space and\%
5253         \string\babelpatterns\space or some patterns will not\%
5254         be taken into account. Reported}%
5255       \fi
5256     \ifx\@empty#1%

```

```

5257     \protected@edef\bbl@patterns@{\bbl@patterns@space#2}%
5258   \else
5259     \edef\bbl@tempb{\zap@space#1 \@empty}%
5260     \bbl@for\bbl@tempa\bbl@tempb{%
5261       \bbl@fixname\bbl@tempa
5262       \bbl@iflanguage\bbl@tempa{%
5263         \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5264           \ifundefined\bbl@patterns@\bbl@tempa}%
5265           \@empty
5266           {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5267           #2}}}%
5268   \fi}}

```

9.5 Southeast Asian scripts

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

```

5269% TODO - to a lua file
5270\directlua{
5271  Babel = Babel or {}
5272  Babel.linebreaking = Babel.linebreaking or {}
5273  Babel.linebreaking.before = {}
5274  Babel.linebreaking.after = {}
5275  Babel.locale = {} % Free to use, indexed by \localeid
5276  function Babel.linebreaking.add_before(func, pos)
5277    tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5278    if pos == nil then
5279      table.insert(Babel.linebreaking.before, func)
5280    else
5281      table.insert(Babel.linebreaking.before, pos, func)
5282    end
5283  end
5284  function Babel.linebreaking.add_after(func)
5285    tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5286    table.insert(Babel.linebreaking.after, func)
5287  end
5288}
5289\def\bbl@intraspace#1 #2 #3\@@{%
5290  \directlua{
5291    Babel = Babel or {}
5292    Babel.intraspaces = Babel.intraspaces or {}
5293    Babel.intraspaces['\csname bbl@sbcpr@language\endcsname'] = %
5294      {b = #1, p = #2, m = #3}
5295    Babel.locale_props[\the\localeid].intraspace = %
5296      {b = #1, p = #2, m = #3}
5297  }}
5298\def\bbl@intrapenalty#1\@@{%
5299  \directlua{
5300    Babel = Babel or {}
5301    Babel.intrapenalties = Babel.intrapenalties or {}
5302    Babel.intrapenalties['\csname bbl@sbcpr@language\endcsname'] = #1
5303    Babel.locale_props[\the\localeid].intrapenalty = #1
5304  }}
5305\begingroup
5306\catcode`\%=12
5307\catcode`\^=14
5308\catcode`\'=12
5309\catcode`\~=12
5310\gdef\bbl@seaintraspace{^
5311  \let\bbl@seaintraspace\relax
5312  \directlua{

```



```

5313 Babel = Babel or {}
5314 Babel.sea_enabled = true
5315 Babel.sea_ranges = Babel.sea_ranges or {}
5316 function Babel.set_chrngs (script, chrng)
5317     local c = 0
5318     for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5319         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5320         c = c + 1
5321     end
5322 end
5323 function Babel.sea_disc_to_space (head)
5324     local sea_ranges = Babel.sea_ranges
5325     local last_char = nil
5326     local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5327     for item in node.traverse(head) do
5328         local i = item.id
5329         if i == node.id'glyph' then
5330             last_char = item
5331         elseif i == 7 and item.subtype == 3 and last_char
5332             and last_char.char > 0x0C99 then
5333             quad = font.getfont(last_char.font).size
5334             for lg, rg in pairs(sea_ranges) do
5335                 if last_char.char > rg[1] and last_char.char < rg[2] then
5336                     lg = lg:sub(1, 4) ^% Remove trailing number of, eg, Cyril
5337                     local intraspace = Babel.intraspaces[lg]
5338                     local intrapenalty = Babel.intrapenalties[lg]
5339                     local n
5340                     if intrapenalty ~= 0 then
5341                         n = node.new(14, 0)      ^% penalty
5342                         n.penalty = intrapenalty
5343                         node.insert_before(head, item, n)
5344                     end
5345                     n = node.new(12, 13)      ^% (glue, spaceskip)
5346                     node.setglue(n, intraspace.b * quad,
5347                                 intraspace.p * quad,
5348                                 intraspace.m * quad)
5349                     node.insert_before(head, item, n)
5350                     node.remove(head, item)
5351                 end
5352             end
5353         end
5354     end
5355 end
5356 }^^
5357 \bbl@luahyphenate}

```

9.6 CJK line breaking

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

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

```

5358 \catcode`\%=14
5359 \gdef\bbl@cjk intraspace{%
5360     \let\bbl@cjk intraspace\relax
5361     \directlua{
5362         Babel = Babel or {}
5363         require('babel-data-cjk.lua')
5364         Babel.cjk_enabled = true
5365         function Babel.cjk_linebreak(head)
5366             local GLYPH = node.id'glyph'

```

```

5367     local last_char = nil
5368     local quad = 655360      % 10 pt = 655360 = 10 * 65536
5369     local last_class = nil
5370     local last_lang = nil
5371
5372     for item in node.traverse(head) do
5373         if item.id == GLYPH then
5374
5375             local lang = item.lang
5376
5377             local LOCALE = node.get_attribute(item,
5378                 Babel.attr_locale)
5379             local props = Babel.locale_props[LOCALE]
5380
5381             local class = Babel.cjk_class[item.char].c
5382
5383             if props.cjk_quotes and props.cjk_quotes[item.char] then
5384                 class = props.cjk_quotes[item.char]
5385             end
5386
5387             if class == 'cp' then class = 'cl' end % ]] as CL
5388             if class == 'id' then class = 'I' end
5389
5390             local br = 0
5391             if class and last_class and Babel.cjk_breaks[last_class][class] then
5392                 br = Babel.cjk_breaks[last_class][class]
5393             end
5394
5395             if br == 1 and props.linebreak == 'c' and
5396                 lang ~= \the\l@nohyphenation\space and
5397                 last_lang ~= \the\l@nohyphenation then
5398                 local intrapenalty = props.intrapenalty
5399                 if intrapenalty ~= 0 then
5400                     local n = node.new(14, 0)      % penalty
5401                     n.penalty = intrapenalty
5402                     node.insert_before(head, item, n)
5403                 end
5404                 local intraspace = props.intraspace
5405                 local n = node.new(12, 13)      % (glue, spaceskip)
5406                 node.setglue(n, intraspace.b * quad,
5407                     intraspace.p * quad,
5408                     intraspace.m * quad)
5409                 node.insert_before(head, item, n)
5410             end
5411
5412             if font.getfont(item.font) then
5413                 quad = font.getfont(item.font).size
5414             end
5415             last_class = class
5416             last_lang = lang
5417             else % if penalty, glue or anything else
5418                 last_class = nil
5419             end
5420         end
5421         lang.hyphenate(head)
5422     end
5423 }%
5424 \bbl@luaohyphenate}
5425 \gdef\bbl@luaohyphenate{%
5426     \let\bbl@luaohyphenate\relax
5427     \directlua{
5428         luatexbase.add_to_callback('hyphenate',
5429             function (head, tail)

```

```

5430     if Babel.linebreaking.before then
5431         for k, func in ipairs(Babel.linebreaking.before) do
5432             func(head)
5433         end
5434     end
5435     if Babel.cjk_enabled then
5436         Babel.cjk_linebreak(head)
5437     end
5438     lang.hyphenate(head)
5439     if Babel.linebreaking.after then
5440         for k, func in ipairs(Babel.linebreaking.after) do
5441             func(head)
5442         end
5443     end
5444     if Babel.sea_enabled then
5445         Babel.sea_disc_to_space(head)
5446     end
5447 end,
5448 'Babel.hyphenate')
5449 }
5450 }
5451 \endgroup
5452 \def\bbl@provide@intraspace{%
5453   \bbl@ifunset{\bbl@intsp@{language}}{%
5454     {\expandafter\ifx\csname\bbl@intsp@{language}\endcsname\@empty\else
5455       \bbl@xin@{c}{\bbl@cl{lnbrk}}}%
5456     \ifin@           % cjk
5457       \bbl@cjk@intraspace
5458       \directlua{
5459         Babel = Babel or {}
5460         Babel.locale_props = Babel.locale_props or {}
5461         Babel.locale_props[\the\localeid].linebreak = 'c'
5462       }%
5463       \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@}%
5464       \ifx\bbl@KVP@intrapenalty\@nnil
5465         \bbl@intrapenalty0\@
5466       \fi
5467     \else           % sea
5468       \bbl@sea@intraspace
5469       \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@}%
5470       \directlua{
5471         Babel = Babel or {}
5472         Babel.sea_ranges = Babel.sea_ranges or {}
5473         Babel.set_chranges('\bbl@cl{sbc}',
5474           '\bbl@cl{chrng}')
5475       }%
5476       \ifx\bbl@KVP@intrapenalty\@nnil
5477         \bbl@intrapenalty0\@
5478       \fi
5479     \fi
5480   \fi
5481   \ifx\bbl@KVP@intrapenalty\@nnil\else
5482     \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
5483   \fi}}

```

9.7 Arabic justification

```

5484 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5485 \def\bblar@chars{%
5486   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5487   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5488   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5489 \def\bblar@elongated{%

```

```

5490 0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5491 063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5492 0649,064A}
5493 \begingroup
5494 \catcode\_ =11 \catcode\:=11
5495 \gdef\bblar@nofswarn{\gdef\msg_warning:nrx##1##2##3{}}
5496 \endgroup
5497 \gdef\bbl@arabicjust{%
5498 \let\bbl@arabicjust\relax
5499 \newattribute\bblar@kashida
5500 \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5501 \bblar@kashida=\z@
5502 \bbl@patchfont{\bbl@parsejalt}}%
5503 \directlua{
5504 Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5505 Babel.arabic.elong_map[\the\localeid] = {}
5506 luatexbase.add_to_callback('post_linebreak_filter',
5507 Babel.arabic.justify, 'Babel.arabic.justify')
5508 luatexbase.add_to_callback('hpack_filter',
5509 Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5510 }%
5511 % Save both node lists to make replacement. TODO. Save also widths to
5512 % make computations
5513 \def\bblar@fetchjalt#1#2#3#4{%
5514 \bbl@exp{\bbl@foreach{#1}}{%
5515 \bbl@ifunset\bblar@JE@##1}%
5516 {\setbox\z@\hbox{^^^200d\char"##1#2}}%
5517 {\setbox\z@\hbox{^^^200d\char"\@nameuse\bblar@JE@##1#2}}%
5518 \directlua{%
5519 local last = nil
5520 for item in node.traverse(tex.box[0].head) do
5521 if item.id == node.id'glyph' and item.char > 0x600 and
5522 not (item.char == 0x200D) then
5523 last = item
5524 end
5525 end
5526 Babel.arabic.#3['##1#4'] = last.char
5527 }}}
5528 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5529 % perhaps other tables (falt?, csw?). What about kaf? And diacritic
5530 % positioning?
5531 \gdef\bbl@parsejalt{%
5532 \ifx\addfontfeature\@undefined\else
5533 \bbl@xin@{/e}{/\bbl@cl{\lnbrk}}%
5534 \ifin@
5535 \directlua{%
5536 if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5537 Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5538 tex.print([\string\csname\space bbl@parsejalti\endcsname])
5539 end
5540 }%
5541 \fi
5542 \fi}
5543 \gdef\bbl@parsejalti{%
5544 \begingroup
5545 \let\bbl@parsejalt\relax % To avoid infinite loop
5546 \edef\bbl@tempb{\fontid\font}%
5547 \bblar@nofswarn
5548 \bblar@fetchjalt\bblar@elongated{}{from}}%
5549 \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
5550 \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
5551 \addfontfeature{RawFeature+=jalt}%
5552 % \@namedef\bblar@JE@0643}{06AA}% todo: catch medial kaf

```

```

5553 \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5554 \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
5555 \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
5556 \directlua{%
5557     for k, v in pairs(Babel.arabic.from) do
5558         if Babel.arabic.dest[k] and
5559             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5560             Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5561                 [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5562         end
5563     end
5564 }%
5565 \endgroup}
5566 %
5567 \begingroup
5568 \catcode`#=11
5569 \catcode`~=11
5570 \directlua{
5571
5572 Babel.arabic = Babel.arabic or {}
5573 Babel.arabic.from = {}
5574 Babel.arabic.dest = {}
5575 Babel.arabic.justify_factor = 0.95
5576 Babel.arabic.justify_enabled = true
5577 Babel.arabic.kashida_limit = -1
5578
5579 function Babel.arabic.justify(head)
5580     if not Babel.arabic.justify_enabled then return head end
5581     for line in node.traverse_id(node.id'hlist', head) do
5582         Babel.arabic.justify_hlist(head, line)
5583     end
5584     return head
5585 end
5586
5587 function Babel.arabic.justify_hbox(head, gc, size, pack)
5588     local has_inf = false
5589     if Babel.arabic.justify_enabled and pack == 'exactly' then
5590         for n in node.traverse_id(12, head) do
5591             if n.stretch_order > 0 then has_inf = true end
5592         end
5593         if not has_inf then
5594             Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5595         end
5596     end
5597     return head
5598 end
5599
5600 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5601     local d, new
5602     local k_list, k_item, pos_inline
5603     local width, width_new, full, k_curr, wt_pos, goal, shift
5604     local subst_done = false
5605     local elong_map = Babel.arabic.elong_map
5606     local cnt
5607     local last_line
5608     local GLYPH = node.id'glyph'
5609     local KASHIDA = Babel.attr_kashida
5610     local LOCALE = Babel.attr_locale
5611
5612     if line == nil then
5613         line = {}
5614         line.glue_sign = 1
5615         line.glue_order = 0

```

```

5616     line.head = head
5617     line.shift = 0
5618     line.width = size
5619 end
5620
5621 % Exclude last line. todo. But-- it discards one-word lines, too!
5622 % ? Look for glue = 12:15
5623 if (line.glue_sign == 1 and line.glue_order == 0) then
5624     elongs = {}      % Stores elongated candidates of each line
5625     k_list = {}      % And all letters with kashida
5626     pos_inline = 0   % Not yet used
5627
5628     for n in node.traverse_id(GLYPH, line.head) do
5629         pos_inline = pos_inline + 1 % To find where it is. Not used.
5630
5631         % Elongated glyphs
5632         if elong_map then
5633             local locale = node.get_attribute(n, LOCALE)
5634             if elong_map[locale] and elong_map[locale][n.font] and
5635                 elong_map[locale][n.font][n.char] then
5636                 table.insert(elongs, {node = n, locale = locale} )
5637                 node.set_attribute(n.prev, KASHIDA, 0)
5638             end
5639         end
5640
5641         % Tatwil
5642         if Babel.kashida_wts then
5643             local k_wt = node.get_attribute(n, KASHIDA)
5644             if k_wt > 0 then % todo. parameter for multi inserts
5645                 table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5646             end
5647         end
5648     end
5649 end % of node.traverse_id
5650
5651 if #elongs == 0 and #k_list == 0 then goto next_line end
5652 full = line.width
5653 shift = line.shift
5654 goal = full * Babel.arabic.justify_factor % A bit crude
5655 width = node.dimensions(line.head) % The 'natural' width
5656
5657 % == Elongated ==
5658 % Original idea taken from 'chickenize'
5659 while (#elongs > 0 and width < goal) do
5660     subst_done = true
5661     local x = #elongs
5662     local curr = elongs[x].node
5663     local oldchar = curr.char
5664     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5665     width = node.dimensions(line.head) % Check if the line is too wide
5666     % Substitute back if the line would be too wide and break:
5667     if width > goal then
5668         curr.char = oldchar
5669         break
5670     end
5671     % If continue, pop the just substituted node from the list:
5672     table.remove(elongs, x)
5673 end
5674
5675 % == Tatwil ==
5676 if #k_list == 0 then goto next_line end
5677
5678 width = node.dimensions(line.head) % The 'natural' width

```

```

5679 k_curr = #k_list % Traverse backwards, from the end
5680 wt_pos = 1
5681
5682 while width < goal do
5683   subst_done = true
5684   k_item = k_list[k_curr].node
5685   if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5686     d = node.copy(k_item)
5687     d.char = 0x0640
5688     line.head, new = node.insert_after(line.head, k_item, d)
5689     width_new = node.dimensions(line.head)
5690     if width > goal or width == width_new then
5691       node.remove(line.head, new) % Better compute before
5692       break
5693     end
5694     width = width_new
5695   end
5696   if k_curr == 1 then
5697     k_curr = #k_list
5698     wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5699   else
5700     k_curr = k_curr - 1
5701   end
5702 end
5703
5704 % Limit the number of tatweel by removing them. Not very efficient,
5705 % but it does the job in a quite predictable way.
5706 if Babel.arabic.kashida_limit > -1 then
5707   cnt = 0
5708   for n in node.traverse_id(GLYPH, line.head) do
5709     if n.char == 0x0640 then
5710       cnt = cnt + 1
5711       if cnt > Babel.arabic.kashida_limit then
5712         node.remove(line.head, n)
5713       end
5714     else
5715       cnt = 0
5716     end
5717   end
5718 end
5719
5720 ::next_line::
5721
5722 % Must take into account marks and ins, see luatex manual.
5723 % Have to be executed only if there are changes. Investigate
5724 % what's going on exactly.
5725 if subst_done and not gc then
5726   d = node.hpack(line.head, full, 'exactly')
5727   d.shift = shift
5728   node.insert_before(head, line, d)
5729   node.remove(head, line)
5730 end
5731 end % if process line
5732 end
5733 }
5734 \endgroup
5735 \fi\fi % Arabic just block

```

9.8 Common stuff

```

5736 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5737 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5738 \DisableBabelHook{babel-fontspec}
5739 <<Font selection>>

```

9.9 Automatic fonts and ids switching

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

```
5740% TODO - to a lua file
5741\directlua{
5742Babel.script_blocks = {
5743  ['dflt'] = {},
5744  ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5745             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5746  ['Armn'] = {{0x0530, 0x058F}},
5747  ['Beng'] = {{0x0980, 0x09FF}},
5748  ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5749  ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5750  ['Cyr'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5751            {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5752  ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5753  ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5754            {0xAB00, 0xAB2F}},
5755  ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5756  % Don't follow strictly Unicode, which places some Coptic letters in
5757  % the 'Greek and Coptic' block
5758  ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5759  ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5760            {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5761            {0xF900, 0FAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5762            {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5763            {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5764            {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5765  ['Hebr'] = {{0x0590, 0x05FF}},
5766  ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5767            {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5768  ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5769  ['Knda'] = {{0x0C80, 0x0CFF}},
5770  ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5771            {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5772            {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5773  ['Lao'] = {{0x0E80, 0x0EFF}},
5774  ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5775            {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5776            {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5777  ['Mahj'] = {{0x1150, 0x117F}},
5778  ['Mlym'] = {{0x0D00, 0x0D7F}},
5779  ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5780  ['Orya'] = {{0x0B00, 0x0B7F}},
5781  ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5782  ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5783  ['Taml'] = {{0x0B80, 0x0BFF}},
5784  ['Telu'] = {{0x0C00, 0x0C7F}},
5785  ['Tfng'] = {{0x2D30, 0x2D7F}},
5786  ['Thai'] = {{0x0E00, 0x0E7F}},
5787  ['Tibt'] = {{0x0F00, 0x0FFF}},
5788  ['Vaii'] = {{0xA500, 0xA63F}},
5789  ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5790 }
5791
5792Babel.script_blocks.Cyrs = Babel.script_blocks.Cyril
5793Babel.script_blocks.Hant = Babel.script_blocks.Hans
```



```

5794 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5795
5796 function Babel.locale_map(head)
5797   if not Babel.locale_mapped then return head end
5798
5799   local LOCALE = Babel.attr_locale
5800   local GLYPH = node.id('glyph')
5801   local inmath = false
5802   local toloc_save
5803   for item in node.traverse(head) do
5804     local toloc
5805     if not inmath and item.id == GLYPH then
5806       % Optimization: build a table with the chars found
5807       if Babel.chr_to_loc[item.char] then
5808         toloc = Babel.chr_to_loc[item.char]
5809       else
5810         for lc, maps in pairs(Babel.loc_to_scr) do
5811           for _, rg in pairs(maps) do
5812             if item.char >= rg[1] and item.char <= rg[2] then
5813               Babel.chr_to_loc[item.char] = lc
5814               toloc = lc
5815               break
5816             end
5817           end
5818         end
5819       end
5820       % Now, take action, but treat composite chars in a different
5821       % fashion, because they 'inherit' the previous locale. Not yet
5822       % optimized.
5823       if not toloc and
5824         (item.char >= 0x0300 and item.char <= 0x036F) or
5825         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5826         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5827         toloc = toloc_save
5828       end
5829       if toloc and Babel.locale_props[toloc] and
5830         Babel.locale_props[toloc].letters and
5831         tex.getcatcode(item.char) \string~= 11 then
5832         toloc = nil
5833       end
5834       if toloc and toloc > -1 then
5835         if Babel.locale_props[toloc].lg then
5836           item.lang = Babel.locale_props[toloc].lg
5837           node.set_attribute(item, LOCALE, toloc)
5838         end
5839         if Babel.locale_props[toloc]['/'..item.font] then
5840           item.font = Babel.locale_props[toloc]['/'..item.font]
5841         end
5842         toloc_save = toloc
5843       end
5844       elseif not inmath and item.id == 7 then % Apply recursively
5845         item.replace = item.replace and Babel.locale_map(item.replace)
5846         item.pre      = item.pre and Babel.locale_map(item.pre)
5847         item.post      = item.post and Babel.locale_map(item.post)
5848       elseif item.id == node.id'math' then
5849         inmath = (item.subtype == 0)
5850       end
5851     end
5852   return head
5853 end
5854 }

```

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

different.

```

5855 \newcommand\babelcharproperty[1]{%
5856   \count@=#1\relax
5857   \ifvmode
5858     \expandafter\babel@chprop
5859   \else
5860     \babel@error{\string\babelcharproperty\space can be used only in\\%
5861               vertical mode (preamble or between paragraphs)}%
5862     {See the manual for futher info}%
5863   \fi}
5864 \newcommand\babel@chprop[3][\the\count@]{%
5865   \@tempcnta=#1\relax
5866   \babel@ifunset{\babel@chprop@#2}%
5867   {\babel@error{No property named '#2'. Allowed values are\\%
5868             direction (bc), mirror (bmg), and linebreak (lb)}%
5869   {See the manual for futher info}}%
5870   {%
5871   \loop
5872     \babel@cs{\babel@chprop@#2}{#3}%
5873     \ifnum\count@<\@tempcnta
5874       \advance\count@\@ne
5875     \repeat}
5876 \def\babel@chprop@direction#1{%
5877   \directlua{
5878     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5879     Babel.characters[\the\count@]['d'] = '#1'
5880   }}
5881 \let\babel@chprop@bc\babel@chprop@direction
5882 \def\babel@chprop@mirror#1{%
5883   \directlua{
5884     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5885     Babel.characters[\the\count@]['m'] = '\number#1'
5886   }}
5887 \let\babel@chprop@bmg\babel@chprop@mirror
5888 \def\babel@chprop@linebreak#1{%
5889   \directlua{
5890     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5891     Babel.cjk_characters[\the\count@]['c'] = '#1'
5892   }}
5893 \let\babel@chprop@lb\babel@chprop@linebreak
5894 \def\babel@chprop@locale#1{%
5895   \directlua{
5896     Babel.chr_to_loc = Babel.chr_to_loc or {}
5897     Babel.chr_to_loc[\the\count@] =
5898       \babel@ifblank{#1}{-1000}{\the\babel@cs{id@#1}}\space
5899   }}

```

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.

```

5900 \directlua{
5901   Babel.nohyphenation = \the\l@nohyphenation
5902 }

```

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

```

5903 \beginngroup

```

```

5904 \catcode`\~ = 12
5905 \catcode`\% = 12
5906 \catcode`\& = 14
5907 \catcode`\| = 12
5908 \gdef\babelprehyphenation{%
5909   \@ifnextchar[{\babel@settransform{0}}{\babel@settransform{0}}{}}
5910 \gdef\babelposthyphenation{%
5911   \@ifnextchar[{\babel@settransform{1}}{\babel@settransform{1}}{}}
5912 \gdef\bbl@settransform#1[#2]#3#4#5{%
5913   \ifcase#1
5914     \bbl@activateprehyphen
5915   \or
5916     \bbl@activateposthyphen
5917   \fi
5918   \begingroup
5919     \def\babeltempa{\bbl@add@list\babeltempb}%
5920     \let\babeltempb\@empty
5921     \def\bbl@tempa{#5}%
5922     \bbl@replace\bbl@tempa{,}{ ,}% TODO. Ugly trick to preserve {}
5923     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{%
5924       \bbl@ifsamestring{##1}{remove}%
5925       {\bbl@add@list\babeltempb{nil}}}%
5926     {\directlua{
5927       local rep = {[##1]=}
5928       rep = rep:gsub('^s*(remove)s*$', 'remove = true')
5929       rep = rep:gsub('^s*(insert)s*', 'insert = true, ')
5930       rep = rep:gsub('(string)s*=%s*([^\s,]*)', Babel.capture_func)
5931       if #1 == 0 or #1 == 2 then
5932         rep = rep:gsub('(space)s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5933           'space = {' .. '%2, %3, %4' .. '}')
5934         rep = rep:gsub('(spacefactor)s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5935           'spacefactor = {' .. '%2, %3, %4' .. '}')
5936         rep = rep:gsub('(kashida)s*=%s*([^\s,]*)', Babel.capture_kashida)
5937       else
5938         rep = rep:gsub(' (no)s*=%s*([^\s,]*)', Babel.capture_func)
5939         rep = rep:gsub(' (pre)s*=%s*([^\s,]*)', Babel.capture_func)
5940         rep = rep:gsub(' (post)s*=%s*([^\s,]*)', Babel.capture_func)
5941       end
5942       tex.print([[string\babeltempa{}} .. rep .. [{}]])
5943     }}}%
5944   \bbl@foreach\babeltempb{%
5945     \bbl@forkv{##1}%
5946     \in{,###1},{,nil,step,data,remove,insert,string,no,pre,&
5947       no,post,penalty,kashida,space,spacefactor},%
5948     \ifin\else
5949       \bbl@error
5950       {Bad option '###1' in a transform.\\&
5951       I'll ignore it but expect more errors}%
5952       {See the manual for further info.}%
5953     \fi}}}%
5954   \let\bbl@kv@attribute\relax
5955   \let\bbl@kv@label\relax
5956   \let\bbl@kv@fonts\@empty
5957   \bbl@forkv{#2}{\bbl@csarg\edef{kv##1}{##2}}%
5958   \ifx\bbl@kv@fonts\@empty\else\bbl@settransform\fi
5959   \ifx\bbl@kv@attribute\relax
5960     \ifx\bbl@kv@label\relax\else
5961       \bbl@exp{\bbl@trim\def{\bbl@kv@fonts{\bbl@kv@fonts}}}%
5962       \bbl@replace\bbl@kv@fonts{ }{ ,}%
5963       \edef\bbl@kv@attribute{\bbl@ATR@\bbl@kv@label @#3\bbl@kv@fonts}%
5964       \count@\z@
5965       \def\bbl@elt##1##2##3{%
5966         \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}%

```

```

5967         {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
5968         {\count@\@ne}&%
5969         {\bbl@error
5970         {Transforms cannot be re-assigned to different\\&%
5971         fonts. The conflict is in '\bbl@kv@label'.\\&%
5972         Apply the same fonts or use a different label}&%
5973         {See the manual for further details.}}}&%
5974     }}&%
5975     \bbl@transfont@list
5976     \ifnum\count@=\z@
5977     \bbl@exp{\global\\bbl@add\\bbl@transfont@list
5978     {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
5979     \fi
5980     \bbl@ifunset{\bbl@kv@attribute}&%
5981     {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
5982     {}&%
5983     \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
5984     \fi
5985 \else
5986     \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
5987 \fi
5988 \directlua{
5989     local lbkr = Babel.linebreaking.replacements[#1]
5990     local u = unicode.utf8
5991     local id, attr, label
5992     if #1 == 0 then
5993         id = \the\csname bbl@id@#3\endcsname\space
5994     else
5995         id = \the\csname l@#3\endcsname\space
5996     end
5997     \ifx\bbl@kv@attribute\relax
5998         attr = -1
5999     \else
6000         attr = luatexbase.registernumber'\bbl@kv@attribute'
6001     \fi
6002     \ifx\bbl@kv@label\relax\else &% Same refs:
6003         label = [==[\bbl@kv@label]==]
6004     \fi
6005     &% Convert pattern:
6006     local patt = string.gsub([==[#4]==], '%s', '')
6007     if #1 == 0 then
6008         patt = string.gsub(patt, '|', ' ')
6009     end
6010     if not u.find(patt, '()', nil, true) then
6011         patt = '()' .. patt .. '()'
6012     end
6013     if #1 == 1 then
6014         patt = string.gsub(patt, '%(%)%^', '^()')
6015         patt = string.gsub(patt, '%$(%)', '()$')
6016     end
6017     patt = u.gsub(patt, '{(.)}',
6018         function (n)
6019             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6020         end)
6021     patt = u.gsub(patt, '{(%x%x%x%x+)}',
6022         function (n)
6023             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6024         end)
6025     lbkr[id] = lbkr[id] or {}
6026     table.insert(lbkr[id],
6027         { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6028     }&%
6029 \endgroup}

```

```

6030 \endgroup
6031 \let\bbl@transfont@list\@empty
6032 \def\bbl@settransfont{%
6033   \global\let\bbl@settransfont\relax % Execute only once
6034   \gdef\bbl@transfont{%
6035     \def\bbl@elt####1####2####3{%
6036       \bbl@ifblank{####3}%
6037       {\count@tw@}% Do nothing if no fonts
6038       {\count@z@
6039         \bbl@vforeach{####3}{%
6040           \def\bbl@tempd{#####1}%
6041           \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6042           \ifx\bbl@tempd\bbl@tempe
6043             \count@ne
6044           \else\ifx\bbl@tempd\bbl@transfam
6045             \count@ne
6046           \fi\fi}%
6047         \ifcase\count@
6048           \bbl@csarg\unsetattribute{ATR####2@####1@####3}%
6049         \or
6050           \bbl@csarg\setattribute{ATR####2@####1@####3}\@ne
6051         \fi}}%
6052     \bbl@transfont@list}%
6053   \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6054   \gdef\bbl@transfam{-unknown-}%
6055   \bbl@foreach\bbl@font@fams{%
6056     \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6057     \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6058     {\xdef\bbl@transfam{##1}}%
6059   {}}}
6060 \DeclareRobustCommand\enablelocaletransform[1]{%
6061   \bbl@ifunset{\bbl@ATR#1@\language @}%
6062   {\bbl@error
6063     {'#1' for '\language' cannot be enabled.\\%
6064     Maybe there is a typo or it's a font-dependent transform}%
6065     {See the manual for further details.}}%
6066   {\bbl@csarg\setattribute{ATR#1@\language @}\@ne}}
6067 \DeclareRobustCommand\disablelocaletransform[1]{%
6068   \bbl@ifunset{\bbl@ATR#1@\language @}%
6069   {\bbl@error
6070     {'#1' for '\language' cannot be disabled.\\%
6071     Maybe there is a typo or it's a font-dependent transform}%
6072     {See the manual for further details.}}%
6073   {\bbl@csarg\unsetattribute{ATR#1@\language @}}}
6074 \def\bbl@activateposthyphen{%
6075   \let\bbl@activateposthyphen\relax
6076   \directlua{
6077     require('babel-transforms.lua')
6078     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6079   }}
6080 \def\bbl@activateprehyphen{%
6081   \let\bbl@activateprehyphen\relax
6082   \directlua{
6083     require('babel-transforms.lua')
6084     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6085   }}

```

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.

```

6086 \newcommand\localeprehyphenation[1]{%

```

```
6087 \directlua{ Babel.string_prehyphenation([=#1==], \the\localeid) }}
```

9.10 Bidi

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

```
6088 \def\bbl@activate@preotf{%
6089 \let\bbl@activate@preotf\relax % only once
6090 \directlua{
6091   Babel = Babel or {}
6092   %
6093   function Babel.pre_otfload_v(head)
6094     if Babel.numbers and Babel.digits_mapped then
6095       head = Babel.numbers(head)
6096     end
6097     if Babel.bidi_enabled then
6098       head = Babel.bidi(head, false, dir)
6099     end
6100     return head
6101   end
6102   %
6103   function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6104     if Babel.numbers and Babel.digits_mapped then
6105       head = Babel.numbers(head)
6106     end
6107     if Babel.bidi_enabled then
6108       head = Babel.bidi(head, false, dir)
6109     end
6110     return head
6111   end
6112   %
6113   luatexbase.add_to_callback('pre_linebreak_filter',
6114     Babel.pre_otfload_v,
6115     'Babel.pre_otfload_v',
6116     luatexbase.priority_in_callback('pre_linebreak_filter',
6117       'luaotfload.node_processor') or nil)
6118   %
6119   luatexbase.add_to_callback('hpack_filter',
6120     Babel.pre_otfload_h,
6121     'Babel.pre_otfload_h',
6122     luatexbase.priority_in_callback('hpack_filter',
6123       'luaotfload.node_processor') or nil)
6124 }}
```

The basic setup. The output is modified at a very low level to set the `\bodydir` to the `\pagedir`. Sadly, we have to deal with boxes in math with basic, so the `\bbl@mathboxdir` hack is activated every math with the package option `bidi=`.

```
6125 \breakafterdirmode=1
6126 \ifnum\bbl@bidimode>\@ne % Any bidi= except default=1
6127 \let\bbl@beforeforeign\leavevmode
6128 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6129 \RequirePackage{luatexbase}
6130 \bbl@activate@preotf
6131 \directlua{
6132   require('babel-data-bidi.lua')
6133   \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6134     require('babel-bidi-basic.lua')
6135   \or
6136     require('babel-bidi-basic-r.lua')
6137   \fi}
6138 \newattribute\bbl@attr@dir
6139 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
```

```

6140 \bbl@exp{\output{\bodydir\pagedir\the\output}}
6141 \fi
6142 \chardef\bbl@thetextdir\z@
6143 \chardef\bbl@thepardir\z@
6144 \def\bbl@getluadir#1{%
6145   \directlua{
6146     if tex.#l_dir == 'TLT' then
6147       tex.sprint('0')
6148     elseif tex.#l_dir == 'TRT' then
6149       tex.sprint('1')
6150     end}}
6151 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6152   \ifcase#3\relax
6153     \ifcase\bbl@getluadir{#1}\relax\else
6154       #2 TLT\relax
6155     \fi
6156   \else
6157     \ifcase\bbl@getluadir{#1}\relax
6158       #2 TRT\relax
6159     \fi
6160   \fi}
6161 % ..00PPTT, with masks 0xC (par dir) and 0x3 (text dir)
6162 \def\bbl@thedir{0}
6163 \def\bbl@textdir#1{%
6164   \bbl@setluadir{text}\textdir{#1}%
6165   \chardef\bbl@thetextdir#1\relax
6166   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6167   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6168 \def\bbl@pardir#1{% Used twice
6169   \bbl@setluadir{par}\pardir{#1}%
6170   \chardef\bbl@thepardir#1\relax}
6171 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}% Used once
6172 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}% Unused
6173 \def\bbl@dirparastext{\pardir\the\textdir\relax}% Used once

```

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

```

6174 \ifnum\bbl@bidimode>\z@ % Any bidi=
6175   \def\bbl@insidemath{0}%
6176   \def\bbl@everymath{\def\bbl@insidemath{1}}
6177   \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6178   \frozen@everymath\expandafter{%
6179     \expandafter\bbl@everymath\the\frozen@everymath}
6180   \frozen@everydisplay\expandafter{%
6181     \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6182   \AtBeginDocument{
6183     \directlua{
6184       function Babel.math_box_dir(head)
6185         if not (token.get_macro('bbl@insidemath') == '0') then
6186           if Babel.hlist_has_bidi(head) then
6187             local d = node.new(node.id'dir')
6188             d.dir = '+TRT'
6189             node.insert_before(head, node.has_glyph(head), d)
6190             for item in node.traverse(head) do
6191               node.set_attribute(item,
6192                 Babel.attr_dir, token.get_macro('bbl@thedir'))
6193             end
6194           end
6195         end
6196       return head
6197     end
6198     luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6199       "Babel.math_box_dir", 0)

```

```
6200 } }%
6201 \fi
```

9.11 Layout

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

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

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

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

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

```
6202 \bbl@trace{Redefinitions for bidi layout}
6203 %
6204 <<(*More package options)>> ≡
6205 \chardef\bbl@eqnpos\z@
6206 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6207 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6208 <</More package options>>
6209 %
6210 \ifnum\bbl@bidimode>\z@ % Any bidi=
6211   \ifx\matheqdirmode\@undefined\else
6212     \matheqdirmode\@ne % A luatex primitive
6213   \fi
6214   \let\bbl@eqnodir\relax
6215   \def\bbl@eqdel{()}
6216   \def\bbl@eqnum{%
6217     {\normalfont\normalcolor
6218       \expandafter\@firstoftwo\bbl@eqdel
6219       \theequation
6220       \expandafter\@secondoftwo\bbl@eqdel}}
6221   \def\bbl@puteqno#1{\eqno\hbox{#1}}
6222   \def\bbl@putleqno#1{\leqno\hbox{#1}}
6223   \def\bbl@eqno@flip#1{%
6224     \ifdim\predisplaysize=-\maxdimen
6225       \eqno
6226       \hb@xt@.01pt{\hb@xt@\displaywidth{\hss{#1}}\hss}%
6227     \else
6228       \leqno\hbox{#1}%
6229     \fi}
6230   \def\bbl@leqno@flip#1{%
6231     \ifdim\predisplaysize=-\maxdimen
6232       \leqno
6233       \hb@xt@.01pt{\hss\hb@xt@\displaywidth{{#1}\hss}}%
6234     \else
6235       \eqno\hbox{#1}%
6236     \fi}
6237   \AtBeginDocument{%
6238     \ifx\bbl@noamsmath\relax\else
6239     \ifx\maketag@@@% undefined % Normal equation, eqnarray
6240       \AddToHook{env/equation/begin}{%
```



```

6241 \ifnum\bb1@thetextdir>\z@
6242 \def\bb1@mathboxdir{\def\bb1@insidemath{1}}%
6243 \let\@eqnnum\bb1@eqnum
6244 \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6245 \chardef\bb1@thetextdir\z@
6246 \bb1@add\normalfont{\bb1@eqnodir}%
6247 \ifcase\bb1@eqnpos
6248 \let\bb1@puteqno\bb1@eqno@flip
6249 \or
6250 \let\bb1@puteqno\bb1@leqno@flip
6251 \fi
6252 \fi}%
6253 \ifnum\bb1@eqnpos=\tw@%else
6254 \def\endequation{\bb1@puteqno{\@eqnnum}$\@ignoretrue}%
6255 \fi
6256 \AddToHook{env/eqnarray/begin}{%
6257 \ifnum\bb1@thetextdir>\z@
6258 \def\bb1@mathboxdir{\def\bb1@insidemath{1}}%
6259 \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6260 \chardef\bb1@thetextdir\z@
6261 \bb1@add\normalfont{\bb1@eqnodir}%
6262 \ifnum\bb1@eqnpos=\@ne
6263 \def\@eqnnum{%
6264 \setbox\z@\hbox{\bb1@eqnum}%
6265 \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6266 \else
6267 \let\@eqnnum\bb1@eqnum
6268 \fi
6269 \fi}
6270 % Hack. YA luatex bug?:
6271 \expandafter\bb1@sreplace\csname\endcsname{\@eqno\kern.001pt$}$%
6272 \else % amstex
6273 \bb1@exp{% Hack to hide maybe undefined conditionals:
6274 \chardef\bb1@eqnpos=0%
6275 \<iftagsleft@>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6276 \ifnum\bb1@eqnpos=\@ne
6277 \let\bb1@ams@lap\hbox
6278 \else
6279 \let\bb1@ams@lap\llap
6280 \fi
6281 \ExplSyntaxOn % Required by \bb1@sreplace with \intertext@
6282 \bb1@sreplace\intertext@{\normalbaselines}%
6283 {\normalbaselines
6284 \ifx\bb1@eqnodir\relax\else\bb1@pardir\@ne\bb1@eqnodir\fi}%
6285 \ExplSyntaxOff
6286 \def\bb1@ams@tagbox#1#2{#1{\bb1@eqnodir#2}}% #1=hbox|@lap|flip
6287 \ifx\bb1@ams@lap\hbox % leqno
6288 \def\bb1@ams@flip#1{%
6289 \hbox to 0.01pt{\hss\hbox to\displaywidth{{#1}\hss}}}%
6290 \else % eqno
6291 \def\bb1@ams@flip#1{%
6292 \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}\hss}}}%
6293 \fi
6294 \def\bb1@ams@preset#1{%
6295 \def\bb1@mathboxdir{\def\bb1@insidemath{1}}%
6296 \ifnum\bb1@thetextdir>\z@
6297 \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6298 \bb1@sreplace\textdef@{\hbox}{\bb1@ams@tagbox\hbox}%
6299 \bb1@sreplace\maketag@@@{\hbox}{\bb1@ams@tagbox#1}%
6300 \fi}%
6301 \ifnum\bb1@eqnpos=\tw@%else
6302 \def\bb1@ams@equation{%
6303 \def\bb1@mathboxdir{\def\bb1@insidemath{1}}%

```

```

6304 \ifnum\bb@thetextdir>\z@
6305 \edef\bb@eqnodir{\noexpand\bb@textdir{\the\bb@thetextdir}}%
6306 \chardef\bb@thetextdir\z@
6307 \bb@add\normalfont{\bb@eqnodir}%
6308 \ifcase\bb@eqnpos
6309 \def\veqno##1##2{\bb@eqno@flip{##1##2}}%
6310 \or
6311 \def\veqno##1##2{\bb@leqno@flip{##1##2}}%
6312 \fi
6313 \fi}%
6314 \AddToHook{env/equation/begin}{\bb@ams@equation}%
6315 \AddToHook{env/equation*/begin}{\bb@ams@equation}%
6316 \fi
6317 \AddToHook{env/cases/begin}{\bb@ams@preset\bb@ams@lap}%
6318 \AddToHook{env/multline/begin}{\bb@ams@preset\hbox}%
6319 \AddToHook{env/gather/begin}{\bb@ams@preset\bb@ams@lap}%
6320 \AddToHook{env/gather*/begin}{\bb@ams@preset\bb@ams@lap}%
6321 \AddToHook{env/align/begin}{\bb@ams@preset\bb@ams@lap}%
6322 \AddToHook{env/align*/begin}{\bb@ams@preset\bb@ams@lap}%
6323 \AddToHook{env/alignat/begin}{\bb@ams@preset\bb@ams@lap}%
6324 \AddToHook{env/alignat*/begin}{\bb@ams@preset\bb@ams@lap}%
6325 \AddToHook{env/eqnalign/begin}{\bb@ams@preset\hbox}%
6326 % Hackish, for proper alignment. Don't ask me why it works!:
6327 \bb@exp{% Avoid a 'visible' conditional
6328 \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\tag*{\<fi>}}%
6329 \\\AddToHook{env/alignat*/end}{\<iftag@>\<else>\\tag*{\<fi>}}%
6330 \AddToHook{env/flalign/begin}{\bb@ams@preset\hbox}%
6331 \AddToHook{env/split/before}{%
6332 \def\bb@mathboxdir{\def\bb@insidemath{1}}%
6333 \ifnum\bb@thetextdir>\z@
6334 \bb@ifsamestring\@currentenv{equation}%
6335 {\ifx\bb@ams@lap\hbox % leqno
6336 \def\bb@ams@flip#1{%
6337 \hbox to 0.01pt{\hbox to\displaywidth{##1}\hss}\hss}%
6338 \else
6339 \def\bb@ams@flip#1{%
6340 \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{##1}}}%
6341 \fi}%
6342 }%
6343 \fi}%
6344 \fi\fi}
6345 \fi
6346 \def\bb@provide@extra#1{%
6347 % == Counters: mapdigits ==
6348 % Native digits
6349 \ifx\bb@KVP@mapdigits\@nnil\else
6350 \bb@ifunset{bb@dgnat@language}{%
6351 {\RequirePackage{luatexbase}}%
6352 \bb@activate@preotf
6353 \directlua{
6354 Babel = Babel or {} %%% -> presets in luababel
6355 Babel.digits_mapped = true
6356 Babel.digits = Babel.digits or {}
6357 Babel.digits[\the\localeid] =
6358 table.pack(string.utfvalue('\bb@cl{dgnat}'))
6359 if not Babel.numbers then
6360 function Babel.numbers(head)
6361 local LOCALE = Babel.attr_locale
6362 local GLYPH = node.id'glyph'
6363 local inmath = false
6364 for item in node.traverse(head) do
6365 if not inmath and item.id == GLYPH then
6366 local temp = node.get_attribute(item, LOCALE)

```

```

6367             if Babel.digits[temp] then
6368                 local chr = item.char
6369                 if chr > 47 and chr < 58 then
6370                     item.char = Babel.digits[temp][chr-47]
6371                 end
6372             end
6373             elseif item.id == node.id'math' then
6374                 inmath = (item.subtype == 0)
6375             end
6376         end
6377         return head
6378     end
6379 end
6380 }}%
6381 \fi
6382 % == transforms ==
6383 \ifx\bb@KVP@transforms\@nnil\else
6384     \def\bb@elt##1##2##3{%
6385         \in@{$transforms.}{$##1}%
6386         \ifin@
6387             \def\bb@tempa{##1}%
6388             \bb@replace\bb@tempa{transforms.}{}%
6389             \bb@carg\bb@transforms{babel\bb@tempa}{##2}{##3}%
6390         \fi}%
6391     \csname bbl@inidata@\language\endcsname
6392     \bb@release@transforms\relax % \relax closes the last item.
6393 \fi}
6394 % Start tabular here:
6395 \def\localerestoredirs{%
6396     \ifcase\bb@thetextdir
6397         \ifnum\textdirection=\z@\else\textdir TLT\fi
6398     \else
6399         \ifnum\textdirection=\@ne\else\textdir TRT\fi
6400     \fi
6401     \ifcase\bb@thepardir
6402         \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6403     \else
6404         \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6405     \fi}
6406 \IfBabelLayout{tabular}%
6407 {\chardef\bb@tabular@mode\tw@}% All RTL
6408 {\IfBabelLayout{notabular}%
6409     {\chardef\bb@tabular@mode\z@}%
6410     {\chardef\bb@tabular@mode\@ne}}% Mixed, with LTR cols
6411 \ifnum\bb@bidimode>\@ne % Any lua bidi= except default=1
6412 \ifcase\bb@tabular@mode\or % 1
6413     \let\bb@parabefore\relax
6414     \AddToHook{para/before}{\bb@parabefore}
6415     \AtBeginDocument{%
6416         \bb@replace\@tabular{$}{$}%
6417         \def\bb@insidemath{0}%
6418         \def\bb@parabefore{\localerestoredirs}}%
6419     \ifnum\bb@tabular@mode=\@ne
6420         \bb@ifunset{@tabclassz}{}%
6421         \bb@exp{% Hide conditionals
6422             \\bb@sreplace\\@tabclassz
6423             {\<ifcase>\\@chnum}%
6424             {\\\localerestoredirs\<ifcase>\\@chnum}}}%
6425         \@ifpackageloaded{colortbl}%
6426         {\bb@sreplace\@classz
6427             {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6428         {\@ifpackageloaded{array}%
6429             {\bb@exp{% Hide conditionals

```

```

6430          \\bbl@sreplace\\@classz
6431          {<ifcase>\\@chnum}%
6432          {bgroup\\localerestoredirs<ifcase>\\@chnum}%
6433          \\bbl@sreplace\\@classz
6434          {\\do@row@strut<fi>}{\\do@row@strut<fi>\\egroup}}}%
6435      {}}%
6436  \fi}%
6437 \or % 2
6438 \let\bbl@parabefore\relax
6439 \AddToHook{para/before}{\bbl@parabefore}%
6440 \AtBeginDocument{%
6441   \ifpackageloaded{colortbl}%
6442     {\bbl@replace@tabular{$}{$%
6443      \def\bbl@insidemath{0}%
6444      \def\bbl@parabefore{\localerestoredirs}}%
6445      \bbl@sreplace@classz
6446      {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}}%
6447   {}}%
6448 \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.

```

6449 \AtBeginDocument{%
6450   \ifpackageloaded{multicol}%
6451     {\toks@expandafter{\multi@column@out}%
6452      \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
6453     {}%
6454   \ifpackageloaded{paracol}%
6455     {\edef\pcol@output{%
6456      \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
6457     {}%
6458 \fi
6459 \ifx\bbl@opt@layout@nnil\endinput\fi % if no layout

```

OMEGA provided a companion to `\mathdir` (`\nextfake`) 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`.

```

6460 \ifnum\bbl@bidimode>\z@ % Any bidi=
6461 \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6462   \bbl@exp{%
6463     \def\\bbl@insidemath{0}%
6464     \mathdir\the\bodydir
6465     #1% Once entered in math, set boxes to restore values
6466     <ifmmode>%
6467       \everyvbox{%
6468         \the\everyvbox
6469         \bodydir\the\bodydir
6470         \mathdir\the\mathdir
6471         \everyhbox{\the\everyhbox}%
6472         \everyvbox{\the\everyvbox}}%
6473       \everyhbox{%
6474         \the\everyhbox
6475         \bodydir\the\bodydir
6476         \mathdir\the\mathdir
6477         \everyhbox{\the\everyhbox}%
6478         \everyvbox{\the\everyvbox}}%
6479       \<fi>}}%
6480 \def\@hangfrom#1{%
6481   \setbox\@tempboxa\hbox{#1}%
6482   \hangindent\wd\@tempboxa
6483   \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6484     \shapemode\@ne

```

```

6485 \fi
6486 \noindent\box\@tempboxa}
6487 \fi
6488 \IfBabelLayout{tabular}
6489 {\let\bbL@OL@tabular\@tabular
6490 \bbL@replace\@tabular{$}\{\bbL@nextfake$}%
6491 \let\bbL@NL@tabular\@tabular
6492 \AtBeginDocument{%
6493 \ifx\bbL@NL@tabular\@tabular\else
6494 \bbL@exp{\in{\bbL@nextfake}\{[@tabular]}}%
6495 \ifin\else
6496 \bbL@replace\@tabular{$}\{\bbL@nextfake$}%
6497 \fi
6498 \let\bbL@NL@tabular\@tabular
6499 \fi}}
6500 {}
6501 \IfBabelLayout{lists}
6502 {\let\bbL@OL@list\list
6503 \bbL@sreplace\list{\parshape}\{\bbL@listparshape}%
6504 \let\bbL@NL@list\list
6505 \def\bbL@listparshape#1#2#3{%
6506 \parshape #1 #2 #3 %
6507 \ifnum\bbL@getluadir{page}=\bbL@getluadir{par}\else
6508 \shapemode\tw@
6509 \fi}}
6510 {}
6511 \IfBabelLayout{graphics}
6512 {\let\bbL@pictresetdir\relax
6513 \def\bbL@pictsetdir#1{%
6514 \ifcase\bbL@thetextdir
6515 \let\bbL@pictresetdir\relax
6516 \else
6517 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6518 \or\textdir TLT
6519 \else\bodydir TLT \textdir TLT
6520 \fi
6521 % \((text|par)dir required in pgf:
6522 \def\bbL@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6523 \fi}%
6524 \AddToHook{env/picture/begin}\{\bbL@pictsetdir\tw@}%
6525 \directlua{
6526 Babel.get_picture_dir = true
6527 Babel.picture_has_bidi = 0
6528 %
6529 function Babel.picture_dir (head)
6530 if not Babel.get_picture_dir then return head end
6531 if Babel.hlist_has_bidi(head) then
6532 Babel.picture_has_bidi = 1
6533 end
6534 return head
6535 end
6536 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6537 "Babel.picture_dir")
6538 }%
6539 \AtBeginDocument{%
6540 \def\LS@rot{%
6541 \setbox\@outputbox\vbox{%
6542 \hbox dir TLT{\rotatebox{90}\{box\@outputbox}}}%
6543 \long\def\put(#1,#2)#3{%
6544 \@killglue
6545 % Try:
6546 \ifx\bbL@pictresetdir\relax
6547 \def\bbL@tempc{0}%

```

```

6548 \else
6549 \directlua{
6550   Babel.get_picture_dir = true
6551   Babel.picture_has_bidi = 0
6552 }%
6553 \setbox\z@\hb@xt@\z@{%
6554   \@defaultunitsset\@tempdimc{#1}\unitlength
6555   \kern\@tempdimc
6556   #3\hss}% TODO: #3 executed twice (below). That's bad.
6557 \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6558 \fi
6559 % Do:
6560 \@defaultunitsset\@tempdimc{#2}\unitlength
6561 \raise\@tempdimc\hb@xt@\z@{%
6562   \@defaultunitsset\@tempdimc{#1}\unitlength
6563   \kern\@tempdimc
6564   {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
6565 \ignorespaces}%
6566 \MakeRobust\put}%
6567 \AtBeginDocument
6568 {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir\@gobble}%
6569 \ifx\pgfpicture\@undefined\else % TODO. Allow deactivate?
6570   \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6571   \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6572   \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6573 \fi
6574 \ifx\tikzpicture\@undefined\else
6575   \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\tw@}%
6576   \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6577   \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
6578 \fi
6579 \ifx\tcolorbox\@undefined\else
6580   \def\tcb@drawing@env@begin{%
6581     \csname tcb@before@tcb@split@state\endcsname
6582     \bbl@pictsetdir\tw@
6583     \begin{\kvtcb@graphenv}%
6584     \tcb@bbdraw%
6585     \tcb@apply@graph@patches
6586   }%
6587   \def\tcb@drawing@env@end{%
6588     \end{\kvtcb@graphenv}%
6589     \bbl@pictresetdir
6590     \csname tcb@after@tcb@split@state\endcsname
6591   }%
6592 \fi
6593 }}
6594 {}

```

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.

```

6595 \IfBabelLayout{counters*}%
6596 {\bbl@add\bbl@opt@layout{.counters.}%
6597 \directlua{
6598   luatexbase.add_to_callback("process_output_buffer",
6599     Babel.discard_sublr , "Babel.discard_sublr") }%
6600 }}
6601 \IfBabelLayout{counters}%
6602 {\let\bbl@0L@@textsuperscript\@textsuperscript
6603 \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6604 \let\bbl@latinarabic=\@arabic
6605 \let\bbl@0L@@arabic\@arabic
6606 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%

```

```

6607 \ifpackagewith{babel}{bidi=default}%
6608 {\let\bbl@asciroman=\@roman
6609 \let\bbl@0L@roman=\@roman
6610 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
6611 \let\bbl@asciRoman=\@Roman
6612 \let\bbl@0L@roman=\@Roman
6613 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciRoman#1}}}%
6614 \let\bbl@0L\labelenumii\labelenumii
6615 \def\labelenumii{}\theenumii}%
6616 \let\bbl@0L@p@enumiii\p@enumiii
6617 \def\p@enumiii{\p@enumii}\theenumii{}\}\}\}\}
6618 <<Footnote changes>>
6619 \IfBabelLayout{footnotes}%
6620 {\let\bbl@0L@footnote\footnote
6621 \BabelFootnote\footnote\languagename{}\}\}%
6622 \BabelFootnote\localfootnote\languagename{}\}\}%
6623 \BabelFootnote\mainfootnote{}\}\}\}\}
6624 {}

```

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.

```

6625 \IfBabelLayout{extras}%
6626 {\bbl@ncarg\let\bbl@0L@underline{underline }%
6627 \bbl@carg\bbl@sreplace{underline }%
6628 {\$@@underline}\bgroup\bbl@nextfake$@@underline}%
6629 \bbl@carg\bbl@sreplace{underline }%
6630 {\m@th$}\m@th$\egroup}%
6631 \let\bbl@0L@LaTeXe\LaTeXe
6632 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6633 \if b\expandafter\@car\f@series\@nil\boldmath\fi
6634 \babelsublr{%
6635 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}%
6636 {}
6637 </luatex>

```

9.12 Lua: transforms

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

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

```

6638 (*transforms)
6639 Babel.linebreaking.replacements = {}
6640 Babel.linebreaking.replacements[0] = {} -- pre
6641 Babel.linebreaking.replacements[1] = {} -- post
6642
6643 -- Discretionaries contain strings as nodes
6644 function Babel.str_to_nodes(fn, matches, base)
6645   local n, head, last
6646   if fn == nil then return nil end
6647   for s in string.utfvalues(fn(matches)) do
6648     if base.id == 7 then
6649       base = base.replace
6650     end
6651     n = node.copy(base)
6652     n.char = s

```

```

6653     if not head then
6654         head = n
6655     else
6656         last.next = n
6657     end
6658     last = n
6659 end
6660 return head
6661 end
6662
6663 Babel.fetch_subtext = {}
6664
6665 Babel.ignore_pre_char = function(node)
6666     return (node.lang == Babel.nohyphenation)
6667 end
6668
6669 -- Merging both functions doesn't seem feasible, because there are too
6670 -- many differences.
6671 Babel.fetch_subtext[0] = function(head)
6672     local word_string = ''
6673     local word_nodes = {}
6674     local lang
6675     local item = head
6676     local inmath = false
6677
6678     while item do
6679
6680         if item.id == 11 then
6681             inmath = (item.subtype == 0)
6682         end
6683
6684         if inmath then
6685             -- pass
6686
6687         elseif item.id == 29 then
6688             local locale = node.get_attribute(item, Babel.attr_locale)
6689
6690             if lang == locale or lang == nil then
6691                 lang = lang or locale
6692                 if Babel.ignore_pre_char(item) then
6693                     word_string = word_string .. Babel.us_char
6694                 else
6695                     word_string = word_string .. unicode.utf8.char(item.char)
6696                 end
6697                 word_nodes[#word_nodes+1] = item
6698             else
6699                 break
6700             end
6701
6702         elseif item.id == 12 and item.subtype == 13 then
6703             word_string = word_string .. ' '
6704             word_nodes[#word_nodes+1] = item
6705
6706             -- Ignore leading unrecognized nodes, too.
6707         elseif word_string ~= '' then
6708             word_string = word_string .. Babel.us_char
6709             word_nodes[#word_nodes+1] = item -- Will be ignored
6710         end
6711
6712         item = item.next
6713     end
6714
6715     -- Here and above we remove some trailing chars but not the

```



```

6716 -- corresponding nodes. But they aren't accessed.
6717 if word_string:sub(-1) == ' ' then
6718     word_string = word_string:sub(1,-2)
6719 end
6720 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6721 return word_string, word_nodes, item, lang
6722 end
6723
6724 Babel.fetch_subtext[1] = function(head)
6725     local word_string = ''
6726     local word_nodes = {}
6727     local lang
6728     local item = head
6729     local inmath = false
6730
6731     while item do
6732
6733         if item.id == 11 then
6734             inmath = (item.subtype == 0)
6735         end
6736
6737         if inmath then
6738             -- pass
6739
6740         elseif item.id == 29 then
6741             if item.lang == lang or lang == nil then
6742                 if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6743                     lang = lang or item.lang
6744                     word_string = word_string .. unicode.utf8.char(item.char)
6745                     word_nodes[#word_nodes+1] = item
6746                 end
6747             else
6748                 break
6749             end
6750
6751         elseif item.id == 7 and item.subtype == 2 then
6752             word_string = word_string .. '='
6753             word_nodes[#word_nodes+1] = item
6754
6755         elseif item.id == 7 and item.subtype == 3 then
6756             word_string = word_string .. '|'
6757             word_nodes[#word_nodes+1] = item
6758
6759         -- (1) Go to next word if nothing was found, and (2) implicitly
6760         -- remove leading USs.
6761         elseif word_string == '' then
6762             -- pass
6763
6764         -- This is the responsible for splitting by words.
6765         elseif (item.id == 12 and item.subtype == 13) then
6766             break
6767
6768         else
6769             word_string = word_string .. Babel.us_char
6770             word_nodes[#word_nodes+1] = item -- Will be ignored
6771         end
6772
6773         item = item.next
6774     end
6775
6776     word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6777     return word_string, word_nodes, item, lang
6778 end

```

```

6779
6780 function Babel.pre_hyphenate_replace(head)
6781   Babel.hyphenate_replace(head, 0)
6782 end
6783
6784 function Babel.post_hyphenate_replace(head)
6785   Babel.hyphenate_replace(head, 1)
6786 end
6787
6788 Babel.us_char = string.char(31)
6789
6790 function Babel.hyphenate_replace(head, mode)
6791   local u = unicode.utf8
6792   local lbkr = Babel.linebreaking.replacements[mode]
6793
6794   local word_head = head
6795
6796   while true do -- for each subtext block
6797
6798     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6799
6800     if Babel.debug then
6801       print()
6802       print((mode == 0) and '@@@<' or '@@@>', w)
6803     end
6804
6805     if nw == nil and w == '' then break end
6806
6807     if not lang then goto next end
6808     if not lbkr[lang] then goto next end
6809
6810     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6811     -- loops are nested.
6812     for k=1, #lbkr[lang] do
6813       local p = lbkr[lang][k].pattern
6814       local r = lbkr[lang][k].replace
6815       local attr = lbkr[lang][k].attr or -1
6816
6817       if Babel.debug then
6818         print('*****', p, mode)
6819       end
6820
6821       -- This variable is set in some cases below to the first *byte*
6822       -- after the match, either as found by u.match (faster) or the
6823       -- computed position based on sc if w has changed.
6824       local last_match = 0
6825       local step = 0
6826
6827       -- For every match.
6828       while true do
6829         if Babel.debug then
6830           print('====')
6831         end
6832         local new -- used when inserting and removing nodes
6833
6834         local matches = { u.match(w, p, last_match) }
6835
6836         if #matches < 2 then break end
6837
6838         -- Get and remove empty captures (with ()'s, which return a
6839         -- number with the position), and keep actual captures
6840         -- (from (...)), if any, in matches.
6841         local first = table.remove(matches, 1)

```

```

6842     local last = table.remove(matches, #matches)
6843     -- Non re-fetched substrings may contain \31, which separates
6844     -- subsubstrings.
6845     if string.find(w:sub(first, last-1), Babel.us_char) then break end
6846
6847     local save_last = last -- with A()BC()D, points to D
6848
6849     -- Fix offsets, from bytes to unicode. Explained above.
6850     first = u.len(w:sub(1, first-1)) + 1
6851     last = u.len(w:sub(1, last-1)) -- now last points to C
6852
6853     -- This loop stores in a small table the nodes
6854     -- corresponding to the pattern. Used by 'data' to provide a
6855     -- predictable behavior with 'insert' (w_nodes is modified on
6856     -- the fly), and also access to 'remove'd nodes.
6857     local sc = first-1 -- Used below, too
6858     local data_nodes = {}
6859
6860     local enabled = true
6861     for q = 1, last-first+1 do
6862         data_nodes[q] = w_nodes[sc+q]
6863         if enabled
6864             and attr > -1
6865             and not node.has_attribute(data_nodes[q], attr)
6866         then
6867             enabled = false
6868         end
6869     end
6870
6871     -- This loop traverses the matched substring and takes the
6872     -- corresponding action stored in the replacement list.
6873     -- sc = the position in substr nodes / string
6874     -- rc = the replacement table index
6875     local rc = 0
6876
6877     while rc < last-first+1 do -- for each replacement
6878         if Babel.debug then
6879             print('.....', rc + 1)
6880         end
6881         sc = sc + 1
6882         rc = rc + 1
6883
6884         if Babel.debug then
6885             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6886             local ss = ''
6887             for itt in node.traverse(head) do
6888                 if itt.id == 29 then
6889                     ss = ss .. unicode.utf8.char(itt.char)
6890                 else
6891                     ss = ss .. '{' .. itt.id .. '}'
6892                 end
6893             end
6894             print('*****', ss)
6895
6896         end
6897
6898         local crep = r[rc]
6899         local item = w_nodes[sc]
6900         local item_base = item
6901         local placeholder = Babel.us_char
6902         local d
6903
6904         if crep and crep.data then

```

```

6905         item_base = data_nodes[crep.data]
6906     end
6907
6908     if crep then
6909         step = crep.step or 0
6910     end
6911
6912     if (not enabled) or (crep and next(crep) == nil) then -- = {}
6913         last_match = save_last    -- Optimization
6914         goto next
6915
6916     elseif crep == nil or crep.remove then
6917         node.remove(head, item)
6918         table.remove(w_nodes, sc)
6919         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6920         sc = sc - 1 -- Nothing has been inserted.
6921         last_match = utf8.offset(w, sc+1+step)
6922         goto next
6923
6924     elseif crep and crep.kashida then -- Experimental
6925         node.set_attribute(item,
6926             Babel.attr_kashida,
6927             crep.kashida)
6928         last_match = utf8.offset(w, sc+1+step)
6929         goto next
6930
6931     elseif crep and crep.string then
6932         local str = crep.string(matches)
6933         if str == '' then -- Gather with nil
6934             node.remove(head, item)
6935             table.remove(w_nodes, sc)
6936             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6937             sc = sc - 1 -- Nothing has been inserted.
6938         else
6939             local loop_first = true
6940             for s in string.utfvalues(str) do
6941                 d = node.copy(item_base)
6942                 d.char = s
6943                 if loop_first then
6944                     loop_first = false
6945                     head, new = node.insert_before(head, item, d)
6946                     if sc == 1 then
6947                         word_head = head
6948                     end
6949                     w_nodes[sc] = d
6950                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6951                 else
6952                     sc = sc + 1
6953                     head, new = node.insert_before(head, item, d)
6954                     table.insert(w_nodes, sc, new)
6955                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6956                 end
6957                 if Babel.debug then
6958                     print('.....', 'str')
6959                     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6960                 end
6961             end -- for
6962             node.remove(head, item)
6963         end -- if ''
6964         last_match = utf8.offset(w, sc+1+step)
6965         goto next
6966
6967     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then

```

```

6968         d = node.new(7, 3) -- (disc, regular)
6969         d.pre      = Babel.str_to_nodes(crep.pre, matches, item_base)
6970         d.post      = Babel.str_to_nodes(crep.post, matches, item_base)
6971         d.replace    = Babel.str_to_nodes(crep.no, matches, item_base)
6972         d.attr = item_base.attr
6973         if crep.pre == nil then -- TeXbook p96
6974             d.penalty = crep.penalty or tex.hyphenpenalty
6975         else
6976             d.penalty = crep.penalty or tex.exhyphenpenalty
6977         end
6978         placeholder = '|'
6979         head, new = node.insert_before(head, item, d)
6980
6981     elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6982         -- ERROR
6983
6984     elseif crep and crep.penalty then
6985         d = node.new(14, 0) -- (penalty, userpenalty)
6986         d.attr = item_base.attr
6987         d.penalty = crep.penalty
6988         head, new = node.insert_before(head, item, d)
6989
6990     elseif crep and crep.space then
6991         -- 655360 = 10 pt = 10 * 65536 sp
6992         d = node.new(12, 13) -- (glue, spaceskip)
6993         local quad = font.getfont(item_base.font).size or 655360
6994         node.setglue(d, crep.space[1] * quad,
6995                       crep.space[2] * quad,
6996                       crep.space[3] * quad)
6997         if mode == 0 then
6998             placeholder = ' '
6999         end
7000         head, new = node.insert_before(head, item, d)
7001
7002     elseif crep and crep.spacefactor then
7003         d = node.new(12, 13) -- (glue, spaceskip)
7004         local base_font = font.getfont(item_base.font)
7005         node.setglue(d,
7006                     crep.spacefactor[1] * base_font.parameters['space'],
7007                     crep.spacefactor[2] * base_font.parameters['space_stretch'],
7008                     crep.spacefactor[3] * base_font.parameters['space_shrink'])
7009         if mode == 0 then
7010             placeholder = ' '
7011         end
7012         head, new = node.insert_before(head, item, d)
7013
7014     elseif mode == 0 and crep and crep.space then
7015         -- ERROR
7016
7017     end -- ie replacement cases
7018
7019     -- Shared by disc, space and penalty.
7020     if sc == 1 then
7021         word_head = head
7022     end
7023     if crep.insert then
7024         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7025         table.insert(w_nodes, sc, new)
7026         last = last + 1
7027     else
7028         w_nodes[sc] = d
7029         node.remove(head, item)
7030         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)

```

```

7031         end
7032
7033         last_match = utf8.offset(w, sc+1+step)
7034
7035         ::next::
7036
7037     end -- for each replacement
7038
7039     if Babel.debug then
7040         print('.....', '/')
7041         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7042     end
7043
7044     end -- for match
7045
7046     end -- for patterns
7047
7048     ::next::
7049     word_head = nw
7050 end -- for substring
7051 return head
7052 end
7053
7054 -- This table stores capture maps, numbered consecutively
7055 Babel.capture_maps = {}
7056
7057 -- The following functions belong to the next macro
7058 function Babel.capture_func(key, cap)
7059     local ret = "[" .. cap:gsub('{{([0-9])}}', ")]..m[%1]..[") .. "]"
7060     local cnt
7061     local u = unicode.utf8
7062     ret, cnt = ret:gsub('{{([0-9])|([^\]]+)|(.-)}}', Babel.capture_func_map)
7063     if cnt == 0 then
7064         ret = u.gsub(ret, '{{(%x%x%x%x+)}}',
7065             function (n)
7066                 return u.char(tonumber(n, 16))
7067             end)
7068     end
7069     ret = ret:gsub("%[%[%]]%.", '')
7070     ret = ret:gsub("%.%[%[%]]%", '')
7071     return key .. [[=function(m) return ]] .. ret .. [[ end]]
7072 end
7073
7074 function Babel.capt_map(from, mapno)
7075     return Babel.capture_maps[mapno][from] or from
7076 end
7077
7078 -- Handle the {n|abc|ABC} syntax in captures
7079 function Babel.capture_func_map(capno, from, to)
7080     local u = unicode.utf8
7081     from = u.gsub(from, '{{(%x%x%x%x+)}}',
7082         function (n)
7083             return u.char(tonumber(n, 16))
7084         end)
7085     to = u.gsub(to, '{{(%x%x%x%x+)}}',
7086         function (n)
7087             return u.char(tonumber(n, 16))
7088         end)
7089     local froms = {}
7090     for s in string.utfcharacters(from) do
7091         table.insert(froms, s)
7092     end
7093     local cnt = 1

```

```

7094 table.insert(Babel.capture_maps, {})
7095 local mlen = table.getn(Babel.capture_maps)
7096 for s in string.utfcharacters(to) do
7097     Babel.capture_maps[mlen][froms[cnt]] = s
7098     cnt = cnt + 1
7099 end
7100 return "]]..Babel.capt_map(m[" .. capno .. "], " ..
7101     (mlen) .. ").." .. "["
7102 end
7103
7104 -- Create/Extend reversed sorted list of kashida weights:
7105 function Babel.capture_kashida(key, wt)
7106     wt = tonumber(wt)
7107     if Babel.kashida_wts then
7108         for p, q in ipairs(Babel.kashida_wts) do
7109             if wt == q then
7110                 break
7111             elseif wt > q then
7112                 table.insert(Babel.kashida_wts, p, wt)
7113                 break
7114             elseif table.getn(Babel.kashida_wts) == p then
7115                 table.insert(Babel.kashida_wts, wt)
7116             end
7117         end
7118     else
7119         Babel.kashida_wts = { wt }
7120     end
7121     return 'kashida = ' .. wt
7122 end
7123
7124 -- Experimental: applies prehyphenation transforms to a string (letters
7125 -- and spaces).
7126 function Babel.string_prehyphenation(str, locale)
7127     local n, head, last, res
7128     head = node.new(8, 0) -- dummy (hack just to start)
7129     last = head
7130     for s in string.utfvalues(str) do
7131         if s == 20 then
7132             n = node.new(12, 0)
7133         else
7134             n = node.new(29, 0)
7135             n.char = s
7136         end
7137         node.set_attribute(n, Babel.attr_locale, locale)
7138         last.next = n
7139         last = n
7140     end
7141     head = Babel.hyphenate_replace(head, 0)
7142     res = ''
7143     for n in node.traverse(head) do
7144         if n.id == 12 then
7145             res = res .. ' '
7146         elseif n.id == 29 then
7147             res = res .. unicode.utf8.char(n.char)
7148         end
7149     end
7150     tex.print(res)
7151 end
7152 </transforms>

```

9.13 Lua: Auto bidi with basic and basic-r

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

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

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

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

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

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

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

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

```
7153 (*basic-r)
7154 Babel = Babel or {}
7155
7156 Babel.bidi_enabled = true
7157
7158 require('babel-data-bidi.lua')
7159
7160 local characters = Babel.characters
7161 local ranges = Babel.ranges
7162
7163 local DIR = node.id("dir")
7164
7165 local function dir_mark(head, from, to, outer)
7166   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
7167   local d = node.new(DIR)
7168   d.dir = '+' .. dir
7169   node.insert_before(head, from, d)
7170   d = node.new(DIR)
7171   d.dir = '-' .. dir
7172   node.insert_after(head, to, d)
7173 end
7174
7175 function Babel.bidi(head, ispar)
7176   local first_n, last_n      -- first and last char with nums
7177   local last_es              -- an auxiliary 'last' used with nums
```



```

7178 local first_d, last_d          -- first and last char in L/R block
7179 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):

```

7180 local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7181 local strong_lr = (strong == 'l') and 'l' or 'r'
7182 local outer = strong
7183
7184 local new_dir = false
7185 local first_dir = false
7186 local inmath = false
7187
7188 local last_lr
7189
7190 local type_n = ''
7191
7192 for item in node.traverse(head) do
7193
7194   -- three cases: glyph, dir, otherwise
7195   if item.id == node.id'glyph'
7196     or (item.id == 7 and item.subtype == 2) then
7197
7198     local itemchar
7199     if item.id == 7 and item.subtype == 2 then
7200       itemchar = item.replace.char
7201     else
7202       itemchar = item.char
7203     end
7204     local chardata = characters[itemchar]
7205     dir = chardata and chardata.d or nil
7206     if not dir then
7207       for nn, et in ipairs(ranges) do
7208         if itemchar < et[1] then
7209           break
7210         elseif itemchar <= et[2] then
7211           dir = et[3]
7212           break
7213         end
7214       end
7215     end
7216     dir = dir or 'l'
7217     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).

```

7218   if new_dir then
7219     attr_dir = 0
7220     for at in node.traverse(item.attr) do
7221       if at.number == Babel.attr_dir then
7222         attr_dir = at.value & 0x3
7223       end
7224     end
7225     if attr_dir == 1 then
7226       strong = 'r'
7227     elseif attr_dir == 2 then
7228       strong = 'al'
7229     else
7230       strong = 'l'
7231     end

```

```

7232     strong_lr = (strong == 'l') and 'l' or 'r'
7233     outer = strong_lr
7234     new_dir = false
7235 end
7236
7237     if dir == 'nsm' then dir = strong end          -- W1

```

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

```

7238     dir_real = dir          -- We need dir_real to set strong below
7239     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:

```

7240     if strong == 'al' then
7241         if dir == 'en' then dir = 'an' end          -- W2
7242         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7243         strong_lr = 'r'          -- W3
7244     end

```

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

```

7245     elseif item.id == node.id'dir' and not inmath then
7246         new_dir = true
7247         dir = nil
7248     elseif item.id == node.id'math' then
7249         inmath = (item.subtype == 0)
7250     else
7251         dir = nil          -- Not a char
7252     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>.

```

7253     if dir == 'en' or dir == 'an' or dir == 'et' then
7254         if dir ~= 'et' then
7255             type_n = dir
7256         end
7257         first_n = first_n or item
7258         last_n = last_es or item
7259         last_es = nil
7260     elseif dir == 'es' and last_n then -- W3+W6
7261         last_es = item
7262     elseif dir == 'cs' then          -- it's right - do nothing
7263     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7264         if strong_lr == 'r' and type_n ~= '' then
7265             dir_mark(head, first_n, last_n, 'r')
7266         elseif strong_lr == 'l' and first_d and type_n == 'an' then
7267             dir_mark(head, first_n, last_n, 'r')
7268             dir_mark(head, first_d, last_d, outer)
7269             first_d, last_d = nil, nil
7270         elseif strong_lr == 'l' and type_n ~= '' then
7271             last_d = last_n
7272         end
7273         type_n = ''
7274         first_n, last_n = nil, nil
7275     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:

```

7276     if dir == 'l' or dir == 'r' then
7277         if dir ~= outer then

```

```

7278     first_d = first_d or item
7279     last_d = item
7280     elseif first_d and dir ~= strong_lr then
7281         dir_mark(head, first_d, last_d, outer)
7282         first_d, last_d = nil, nil
7283     end
7284 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.

```

7285     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7286         item.char = characters[item.char] and
7287             characters[item.char].m or item.char
7288     elseif (dir or new_dir) and last_lr ~= item then
7289         local mir = outer .. strong_lr .. (dir or outer)
7290         if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7291             for ch in node.traverse(node.next(last_lr)) do
7292                 if ch == item then break end
7293                 if ch.id == node.id'glyph' and characters[ch.char] then
7294                     ch.char = characters[ch.char].m or ch.char
7295                 end
7296             end
7297         end
7298     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).

```

7299     if dir == 'l' or dir == 'r' then
7300         last_lr = item
7301         strong = dir_real      -- Don't search back - best save now
7302         strong_lr = (strong == 'l') and 'l' or 'r'
7303     elseif new_dir then
7304         last_lr = nil
7305     end
7306 end

```

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

```

7307     if last_lr and outer == 'r' then
7308         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7309             if characters[ch.char] then
7310                 ch.char = characters[ch.char].m or ch.char
7311             end
7312         end
7313     end
7314     if first_n then
7315         dir_mark(head, first_n, last_n, outer)
7316     end
7317     if first_d then
7318         dir_mark(head, first_d, last_d, outer)
7319     end

```

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

```

7320     return node.prev(head) or head
7321 end
7322 </basic-r>

```

And here the Lua code for bidi=basic:

```

7323 <(*basic>
7324 Babel = Babel or {}
7325

```

```

7326 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
7327
7328 Babel.fontmap = Babel.fontmap or {}
7329 Babel.fontmap[0] = {}      -- l
7330 Babel.fontmap[1] = {}      -- r
7331 Babel.fontmap[2] = {}      -- al/an
7332
7333 Babel.bidi_enabled = true
7334 Babel.mirroring_enabled = true
7335
7336 require('babel-data-bidi.lua')
7337
7338 local characters = Babel.characters
7339 local ranges = Babel.ranges
7340
7341 local DIR = node.id('dir')
7342 local GLYPH = node.id('glyph')
7343
7344 local function insert_implicit(head, state, outer)
7345   local new_state = state
7346   if state.sim and state.eim and state.sim ~= state.eim then
7347     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
7348     local d = node.new(DIR)
7349     d.dir = '+' .. dir
7350     node.insert_before(head, state.sim, d)
7351     local d = node.new(DIR)
7352     d.dir = '-' .. dir
7353     node.insert_after(head, state.eim, d)
7354   end
7355   new_state.sim, new_state.eim = nil, nil
7356   return head, new_state
7357 end
7358
7359 local function insert_numeric(head, state)
7360   local new
7361   local new_state = state
7362   if state.san and state.ean and state.san ~= state.ean then
7363     local d = node.new(DIR)
7364     d.dir = '+TLT'
7365     _, new = node.insert_before(head, state.san, d)
7366     if state.san == state.sim then state.sim = new end
7367     local d = node.new(DIR)
7368     d.dir = '-TLT'
7369     _, new = node.insert_after(head, state.ean, d)
7370     if state.ean == state.eim then state.eim = new end
7371   end
7372   new_state.san, new_state.ean = nil, nil
7373   return head, new_state
7374 end
7375
7376 -- TODO - \hbox with an explicit dir can lead to wrong results
7377 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7378 -- was s made to improve the situation, but the problem is the 3-dir
7379 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7380 -- well.
7381
7382 function Babel.bidi(head, ispar, hdir)
7383   local d      -- d is used mainly for computations in a loop
7384   local prev_d = ''
7385   local new_d = false
7386
7387   local nodes = {}
7388   local outer_first = nil

```

```

7389 local inmath = false
7390
7391 local glue_d = nil
7392 local glue_i = nil
7393
7394 local has_en = false
7395 local first_et = nil
7396
7397 local has_hyperlink = false
7398
7399 local ATDIR = Babel.attr_dir
7400
7401 local save_outer
7402 local temp = node.get_attribute(head, ATDIR)
7403 if temp then
7404     temp = temp & 0x3
7405     save_outer = (temp == 0 and 'l') or
7406                  (temp == 1 and 'r') or
7407                  (temp == 2 and 'al')
7408 elseif ispar then -- Or error? Shouldn't happen
7409     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7410 else -- Or error? Shouldn't happen
7411     save_outer = ('TRT' == hdir) and 'r' or 'l'
7412 end
7413 -- when the callback is called, we are just _after_ the box,
7414 -- and the textdir is that of the surrounding text
7415 -- if not ispar and hdir ~= tex.textdir then
7416 --     save_outer = ('TRT' == hdir) and 'r' or 'l'
7417 -- end
7418 local outer = save_outer
7419 local last = outer
7420 -- 'al' is only taken into account in the first, current loop
7421 if save_outer == 'al' then save_outer = 'r' end
7422
7423 local fontmap = Babel.fontmap
7424
7425 for item in node.traverse(head) do
7426
7427     -- In what follows, #node is the last (previous) node, because the
7428     -- current one is not added until we start processing the neutrals.
7429
7430     -- three cases: glyph, dir, otherwise
7431     if item.id == GLYPH
7432        or (item.id == 7 and item.subtype == 2) then
7433
7434         local d_font = nil
7435         local item_r
7436         if item.id == 7 and item.subtype == 2 then
7437             item_r = item.replace -- automatic discs have just 1 glyph
7438         else
7439             item_r = item
7440         end
7441         local chardata = characters[item_r.char]
7442         d = chardata and chardata.d or nil
7443         if not d or d == 'nsm' then
7444             for nn, et in ipairs(ranges) do
7445                 if item_r.char < et[1] then
7446                     break
7447                 elseif item_r.char <= et[2] then
7448                     if not d then d = et[3]
7449                     elseif d == 'nsm' then d_font = et[3]
7450                     end
7451                     break

```

```

7452         end
7453     end
7454 end
7455 d = d or 'l'
7456
7457 -- A short 'pause' in bidi for mapfont
7458 d_font = d_font or d
7459 d_font = (d_font == 'l' and 0) or
7460           (d_font == 'nsm' and 0) or
7461           (d_font == 'r' and 1) or
7462           (d_font == 'al' and 2) or
7463           (d_font == 'an' and 2) or nil
7464 if d_font and fontmap and fontmap[d_font][item_r.font] then
7465     item_r.font = fontmap[d_font][item_r.font]
7466 end
7467
7468 if new_d then
7469     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7470     if inmath then
7471         attr_d = 0
7472     else
7473         attr_d = node.get_attribute(item, ATDIR)
7474         attr_d = attr_d & 0x3
7475     end
7476     if attr_d == 1 then
7477         outer_first = 'r'
7478         last = 'r'
7479     elseif attr_d == 2 then
7480         outer_first = 'r'
7481         last = 'al'
7482     else
7483         outer_first = 'l'
7484         last = 'l'
7485     end
7486     outer = last
7487     has_en = false
7488     first_et = nil
7489     new_d = false
7490 end
7491
7492 if glue_d then
7493     if (d == 'l' and 'l' or 'r') ~= glue_d then
7494         table.insert(nodes, {glue_i, 'on', nil})
7495     end
7496     glue_d = nil
7497     glue_i = nil
7498 end
7499
7500 elseif item.id == DIR then
7501     d = nil
7502
7503     if head ~= item then new_d = true end
7504
7505 elseif item.id == node.id'glue' and item.subtype == 13 then
7506     glue_d = d
7507     glue_i = item
7508     d = nil
7509
7510 elseif item.id == node.id'math' then
7511     inmath = (item.subtype == 0)
7512
7513 elseif item.id == 8 and item.subtype == 19 then
7514     has_hyperlink = true

```

```

7515
7516     else
7517         d = nil
7518     end
7519
7520     -- AL <= EN/ET/ES      -- W2 + W3 + W6
7521     if last == 'al' and d == 'en' then
7522         d = 'an'          -- W3
7523     elseif last == 'al' and (d == 'et' or d == 'es') then
7524         d = 'on'          -- W6
7525     end
7526
7527     -- EN + CS/ES + EN      -- W4
7528     if d == 'en' and #nodes >= 2 then
7529         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7530             and nodes[#nodes-1][2] == 'en' then
7531             nodes[#nodes][2] = 'en'
7532         end
7533     end
7534
7535     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
7536     if d == 'an' and #nodes >= 2 then
7537         if (nodes[#nodes][2] == 'cs')
7538             and nodes[#nodes-1][2] == 'an' then
7539             nodes[#nodes][2] = 'an'
7540         end
7541     end
7542
7543     -- ET/EN                  -- W5 + W7->l / W6->on
7544     if d == 'et' then
7545         first_et = first_et or (#nodes + 1)
7546     elseif d == 'en' then
7547         has_en = true
7548         first_et = first_et or (#nodes + 1)
7549     elseif first_et then      -- d may be nil here !
7550         if has_en then
7551             if last == 'l' then
7552                 temp = 'l'    -- W7
7553             else
7554                 temp = 'en'   -- W5
7555             end
7556         else
7557             temp = 'on'       -- W6
7558         end
7559         for e = first_et, #nodes do
7560             if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7561         end
7562         first_et = nil
7563         has_en = false
7564     end
7565
7566     -- Force mathdir in math if ON (currently works as expected only
7567     -- with 'l')
7568     if inmath and d == 'on' then
7569         d = ('TRT' == tex.mathdir) and 'r' or 'l'
7570     end
7571
7572     if d then
7573         if d == 'al' then
7574             d = 'r'
7575             last = 'al'
7576         elseif d == 'l' or d == 'r' then
7577             last = d

```

```

7578     end
7579     prev_d = d
7580     table.insert(nodes, {item, d, outer_first})
7581 end
7582
7583     outer_first = nil
7584
7585 end
7586
7587 -- TODO -- repeated here in case EN/ET is the last node. Find a
7588 -- better way of doing things:
7589 if first_et then      -- dir may be nil here !
7590     if has_en then
7591         if last == 'l' then
7592             temp = 'l'      -- W7
7593         else
7594             temp = 'en'     -- W5
7595         end
7596     else
7597         temp = 'on'        -- W6
7598     end
7599     for e = first_et, #nodes do
7600         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7601     end
7602 end
7603
7604 -- dummy node, to close things
7605 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7606
7607 ----- NEUTRAL -----
7608
7609 outer = save_outer
7610 last = outer
7611
7612 local first_on = nil
7613
7614 for q = 1, #nodes do
7615     local item
7616
7617     local outer_first = nodes[q][3]
7618     outer = outer_first or outer
7619     last = outer_first or last
7620
7621     local d = nodes[q][2]
7622     if d == 'an' or d == 'en' then d = 'r' end
7623     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7624
7625     if d == 'on' then
7626         first_on = first_on or q
7627     elseif first_on then
7628         if last == d then
7629             temp = d
7630         else
7631             temp = outer
7632         end
7633         for r = first_on, q - 1 do
7634             nodes[r][2] = temp
7635             item = nodes[r][1]      -- MIRRORING
7636             if Babel.mirroring_enabled and item.id == GLYPH
7637                 and temp == 'r' and characters[item.char] then
7638                 local font_mode = ''
7639                 if item.font > 0 and font.fonts[item.font].properties then
7640                     font_mode = font.fonts[item.font].properties.mode

```



```

7641         end
7642         if font_mode ~= 'harf' and font_mode ~= 'plug' then
7643             item.char = characters[item.char].m or item.char
7644         end
7645     end
7646 end
7647     first_on = nil
7648 end
7649
7650     if d == 'r' or d == 'l' then last = d end
7651 end
7652
7653 ----- IMPLICIT, REORDER -----
7654
7655 outer = save_outer
7656 last = outer
7657
7658 local state = {}
7659 state.has_r = false
7660
7661 for q = 1, #nodes do
7662     local item = nodes[q][1]
7663
7664     outer = nodes[q][3] or outer
7665
7666     local d = nodes[q][2]
7667
7668     if d == 'nsm' then d = last end          -- W1
7669     if d == 'en' then d = 'an' end
7670     local isdir = (d == 'r' or d == 'l')
7671
7672     if outer == 'l' and d == 'an' then
7673         state.san = state.san or item
7674         state.ean = item
7675     elseif state.san then
7676         head, state = insert_numeric(head, state)
7677     end
7678
7679     if outer == 'l' then
7680         if d == 'an' or d == 'r' then      -- im -> implicit
7681             if d == 'r' then state.has_r = true end
7682             state.sim = state.sim or item
7683             state.eim = item
7684         elseif d == 'l' and state.sim and state.has_r then
7685             head, state = insert_implicit(head, state, outer)
7686         elseif d == 'l' then
7687             state.sim, state.eim, state.has_r = nil, nil, false
7688         end
7689     else
7690         if d == 'an' or d == 'l' then
7691             if nodes[q][3] then -- nil except after an explicit dir
7692                 state.sim = item -- so we move sim 'inside' the group
7693             else
7694                 state.sim = state.sim or item
7695             end
7696             state.eim = item
7697         elseif d == 'r' and state.sim then
7698             head, state = insert_implicit(head, state, outer)
7699         elseif d == 'r' then
7700             state.sim, state.eim = nil, nil
7701         end
7702     end
7703 end

```

```

7704
7705     if isdir then
7706         last = d           -- Don't search back - best save now
7707     elseif d == 'on' and state.san then
7708         state.san = state.san or item
7709         state.ean = item
7710     end
7711
7712 end
7713
7714 head = node.prev(head) or head
7715
7716 ----- FIX HYPERLINKS -----
7717
7718 if has_hyperlink then
7719     local flag, linking = 0, 0
7720     for item in node.traverse(head) do
7721         if item.id == DIR then
7722             if item.dir == '+TRT' or item.dir == '+TLT' then
7723                 flag = flag + 1
7724             elseif item.dir == '-TRT' or item.dir == '-TLT' then
7725                 flag = flag - 1
7726             end
7727             elseif item.id == 8 and item.subtype == 19 then
7728                 linking = flag
7729             elseif item.id == 8 and item.subtype == 20 then
7730                 if linking > 0 then
7731                     if item.prev.id == DIR and
7732                         (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
7733                         d = node.new(DIR)
7734                         d.dir = item.prev.dir
7735                         node.remove(head, item.prev)
7736                         node.insert_after(head, item, d)
7737                     end
7738                 end
7739                 linking = 0
7740             end
7741         end
7742     end
7743
7744     return head
7745 end
7746 </basic>

```

10 Data for CJK

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

```

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

```

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

11 The ‘nil’ language

This ‘language’ does nothing, except setting the hyphenation patterns to nohyphenation. For this language currently no special definitions are needed or available.

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

```
7747 ⟨*nil⟩
7748 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ v⟨⟨version⟩⟩ Nil language]
7749 \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.

```
7750 \ifx\l@nil\@undefined
7751   \newlanguage\l@nil
7752   \@namedef{bbl@hyphendata@the\l@nil}{}}}% Remove warning
7753   \let\bbl@elt\relax
7754   \edef\bbl@languages{% Add it to the list of languages
7755     \bbl@languages\bbl@elt{nil}{the\l@nil}{}}}%
7756 \fi
```

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

```
7757 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
\datenil
7758 \let\captionnil\@empty
7759 \let\datenil\@empty
```

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

```
7760 \def\bbl@inidata@nil{%
7761   \bbl@elt{identification}{tag.ini}{und}%
7762   \bbl@elt{identification}{load.level}{0}%
7763   \bbl@elt{identification}{charset}{utf8}%
7764   \bbl@elt{identification}{version}{1.0}%
7765   \bbl@elt{identification}{date}{2022-05-16}%
7766   \bbl@elt{identification}{name.local}{nil}%
7767   \bbl@elt{identification}{name.english}{nil}%
7768   \bbl@elt{identification}{name.babel}{nil}%
7769   \bbl@elt{identification}{tag.bcp47}{und}%
7770   \bbl@elt{identification}{language.tag.bcp47}{und}%
7771   \bbl@elt{identification}{tag.opentype}{dflt}%
7772   \bbl@elt{identification}{script.name}{Latin}%
7773   \bbl@elt{identification}{script.tag.bcp47}{Latn}%
7774   \bbl@elt{identification}{script.tag.opentype}{DFLT}%
7775   \bbl@elt{identification}{level}{1}%
7776   \bbl@elt{identification}{encodings}{}%
7777   \bbl@elt{identification}{derivate}{no}}
7778 \@namedef{bbl@tbc@nil}{und}
7779 \@namedef{bbl@lbc@nil}{und}
7780 \@namedef{bbl@casing@nil}{und} % TODO
7781 \@namedef{bbl@lotf@nil}{dflt}
7782 \@namedef{bbl@elname@nil}{nil}
7783 \@namedef{bbl@lname@nil}{nil}
7784 \@namedef{bbl@esname@nil}{Latin}
7785 \@namedef{bbl@sname@nil}{Latin}
7786 \@namedef{bbl@sbc@nil}{Latn}
7787 \@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.

```
7788 \ldf@finish{nil}
7789 ⟨/nil⟩
```

12 Calendars

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

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

```
7790 <<Compute Julian day>> ≡
7791 \def\bbl@fpmo#1#2{(#1-#2*floor(#1/#2))}
7792 \def\bbl@cs@gregleap#1{%
7793   (\bbl@fpmo{#1}{4} == 0) &&
7794   (!((\bbl@fpmo{#1}{100} == 0) && (\bbl@fpmo{#1}{400} != 0)))}
7795 \def\bbl@cs@jd#1#2#3{% year, month, day
7796   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
7797     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
7798     floor((#1 - 1) / 400) + floor((((367 * #2) - 362) / 12) +
7799     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3)} }
7800 <</Compute Julian day>>
```

12.1 Islamic

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

```
7801 <*ca-islamic>
7802 \ExplSyntaxOn
7803 <<Compute Julian day>>
7804 % == islamic (default)
7805 % Not yet implemented
7806 \def\bbl@ca@islamic#1-#2-#3\@@#4#5#6{}
```

The Civil calendar:

```
7807 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
7808   ((#3 + ceil(29.5 * (#2 - 1)) +
7809     (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
7810     1948439.5) - 1) }
7811 \@namedef{\bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
7812 \@namedef{\bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
7813 \@namedef{\bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
7814 \@namedef{\bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
7815 \@namedef{\bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
7816 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@@#5#6#7{%
7817   \edef\bbl@tempa{%
7818     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
7819   \edef#5{%
7820     \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
7821   \edef#6{\fp_eval:n{
7822     min(12, ceil((\bbl@tempa - (29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
7823   \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).

```
7824 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
7825 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
7826 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
7827 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
7828 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
7829 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
7830 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
7831 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
7832 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
7833 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
7834 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
7835 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
```

```

7836 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
7837 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
7838 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
7839 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
7840 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
7841 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
7842 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
7843 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
7844 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
7845 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
7846 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
7847 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
7848 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
7849 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
7850 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
7851 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
7852 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
7853 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
7854 65401,65431,65460,65490,65520}
7855 \namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
7856 \namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
7857 \namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
7858 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
7859 \ifnum#2>2014 \ifnum#2<2038
7860 \bbl@afterfi\expandafter\@gobble
7861 \fi\fi
7862 {\bbl@error{Year~out~of~range}{The~allowed~range~is~2014-2038}}%
7863 \def\bbl@tempd{\fp_eval:n{ % (Julian) day
7864 \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
7865 \count@\@ne
7866 \bbl@foreach\bbl@cs@umalqura@data{%
7867 \advance\count@\@ne
7868 \ifnum##1>\bbl@tempd\else
7869 \edef\bbl@tempe{\the\count@}%
7870 \edef\bbl@tempb{##1}%
7871 \fi}%
7872 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
7873 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1 ) / 12) }}% annus
7874 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
7875 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
7876 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}
7877 \ExplSyntaxOff
7878 \bbl@add\bbl@precalendar{%
7879 \bbl@replace\bbl@ld@calendar{-civil}{}%
7880 \bbl@replace\bbl@ld@calendar{-umalqura}{}%
7881 \bbl@replace\bbl@ld@calendar{+}{}%
7882 \bbl@replace\bbl@ld@calendar{-}{}}
7883 \ca-islamic)

```

12.2 Hebrew

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

```

7884 (*ca-hebrew)
7885 \newcount\bbl@cntcommon
7886 \def\bbl@remainder#1#2#3{%
7887 #3=#1\relax
7888 \divide #3 by #2\relax
7889 \multiply #3 by -#2\relax
7890 \advance #3 by #1\relax}%
7891 \newif\ifbbl@divisible
7892 \def\bbl@checkifdivisible#1#2{%

```

```

7893 {\countdef\tmp=0
7894 \bbl@remainder{#1}{#2}{\tmp}%
7895 \ifnum \tmp=0
7896 \global\bbl@divisibletrue
7897 \else
7898 \global\bbl@divisiblefalse
7899 \fi}}
7900 \newif\ifbbl@gregleap
7901 \def\bbl@ifgregleap#1{%
7902 \bbl@checkifdivisible{#1}{4}%
7903 \ifbbl@divisible
7904 \bbl@checkifdivisible{#1}{100}%
7905 \ifbbl@divisible
7906 \bbl@checkifdivisible{#1}{400}%
7907 \ifbbl@divisible
7908 \bbl@gregleaptrue
7909 \else
7910 \bbl@gregleapfalse
7911 \fi
7912 \else
7913 \bbl@gregleaptrue
7914 \fi
7915 \else
7916 \bbl@gregleapfalse
7917 \fi
7918 \ifbbl@gregleap}
7919 \def\bbl@gregdayspriormonths#1#2#3{%
7920 {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
7921 181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
7922 \bbl@ifgregleap{#2}%
7923 \ifnum #1 > 2
7924 \advance #3 by 1
7925 \fi
7926 \fi
7927 \global\bbl@cntcommon=#3}%
7928 #3=\bbl@cntcommon}
7929 \def\bbl@gregdaysprioryears#1#2{%
7930 {\countdef\tmpc=4
7931 \countdef\tmpb=2
7932 \tmpb=#1\relax
7933 \advance \tmpb by -1
7934 \tmpc=\tmpb
7935 \multiply \tmpc by 365
7936 #2=\tmpc
7937 \tmpc=\tmpb
7938 \divide \tmpc by 4
7939 \advance #2 by \tmpc
7940 \tmpc=\tmpb
7941 \divide \tmpc by 100
7942 \advance #2 by -\tmpc
7943 \tmpc=\tmpb
7944 \divide \tmpc by 400
7945 \advance #2 by \tmpc
7946 \global\bbl@cntcommon=#2\relax}%
7947 #2=\bbl@cntcommon}
7948 \def\bbl@absfromgreg#1#2#3#4{%
7949 {\countdef\tmpd=0
7950 #4=#1\relax
7951 \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
7952 \advance #4 by \tmpd
7953 \bbl@gregdaysprioryears{#3}{\tmpd}%
7954 \advance #4 by \tmpd
7955 \global\bbl@cntcommon=#4\relax}%

```

```

7956 #4=\bbl@cntcommon}
7957 \newif\ifbbl@hebrleap
7958 \def\bbl@checkleaphebrewyear#1{%
7959 {\countdef\tmpa=0
7960 \countdef\tmpb=1
7961 \tmpa=#1\relax
7962 \multiply \tmpa by 7
7963 \advance \tmpa by 1
7964 \bbl@remainder{\tmpa}{19}{\tmpb}%
7965 \ifnum \tmpb < 7
7966 \global\bbl@hebrleaptrue
7967 \else
7968 \global\bbl@hebrleapfalse
7969 \fi}}
7970 \def\bbl@hebreleapsedmonths#1#2{%
7971 {\countdef\tmpa=0
7972 \countdef\tmpb=1
7973 \countdef\tmpc=2
7974 \tmpa=#1\relax
7975 \advance \tmpa by -1
7976 #2=\tmpa
7977 \divide #2 by 19
7978 \multiply #2 by 235
7979 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
7980 \tmpc=\tmpb
7981 \multiply \tmpb by 12
7982 \advance #2 by \tmpb
7983 \multiply \tmpc by 7
7984 \advance \tmpc by 1
7985 \divide \tmpc by 19
7986 \advance #2 by \tmpc
7987 \global\bbl@cntcommon=#2}%
7988 #2=\bbl@cntcommon}
7989 \def\bbl@hebreleapseddays#1#2{%
7990 {\countdef\tmpa=0
7991 \countdef\tmpb=1
7992 \countdef\tmpc=2
7993 \bbl@hebreleapsedmonths{#1}{#2}%
7994 \tmpa=#2\relax
7995 \multiply \tmpa by 13753
7996 \advance \tmpa by 5604
7997 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
7998 \divide \tmpa by 25920
7999 \multiply #2 by 29
8000 \advance #2 by 1
8001 \advance #2 by \tmpa
8002 \bbl@remainder{#2}{7}{\tmpa}%
8003 \ifnum \tmpc < 19440
8004 \ifnum \tmpc < 9924
8005 \else
8006 \ifnum \tmpa=2
8007 \bbl@checkleaphebrewyear{#1}% of a common year
8008 \ifbbl@hebrleap
8009 \else
8010 \advance #2 by 1
8011 \fi
8012 \fi
8013 \fi
8014 \ifnum \tmpc < 16789
8015 \else
8016 \ifnum \tmpa=1
8017 \advance #1 by -1
8018 \bbl@checkleaphebrewyear{#1}% at the end of leap year

```

```

8019             \ifbbl@hebrleap
8020             \advance #2 by 1
8021         \fi
8022     \fi
8023 \fi
8024 \else
8025     \advance #2 by 1
8026 \fi
8027 \bbl@remainder{#2}{7}{\tmpa}%
8028 \ifnum \tmpa=0
8029     \advance #2 by 1
8030 \else
8031     \ifnum \tmpa=3
8032         \advance #2 by 1
8033     \else
8034         \ifnum \tmpa=5
8035             \advance #2 by 1
8036         \fi
8037     \fi
8038 \fi
8039 \global\bbl@cntcommon=#2\relax}%
8040 #2=\bbl@cntcommon}
8041 \def\bbl@daysinhebryear#1#2{%
8042     {\countdef\tmpe=12
8043     \bbl@hebreleapseddays{#1}{\tmpe}%
8044     \advance #1 by 1
8045     \bbl@hebreleapseddays{#1}{#2}%
8046     \advance #2 by -\tmpe
8047     \global\bbl@cntcommon=#2}%
8048 #2=\bbl@cntcommon}
8049 \def\bbl@hebrdayspriormonths#1#2#3{%
8050     {\countdef\tmpf= 14
8051     #3=\ifcase #1\relax
8052         0 \or
8053         0 \or
8054         30 \or
8055         59 \or
8056         89 \or
8057         118 \or
8058         148 \or
8059         148 \or
8060         177 \or
8061         207 \or
8062         236 \or
8063         266 \or
8064         295 \or
8065         325 \or
8066         400
8067     \fi
8068     \bbl@checkleaphebryear{#2}%
8069     \ifbbl@hebrleap
8070         \ifnum #1 > 6
8071             \advance #3 by 30
8072         \fi
8073     \fi
8074     \bbl@daysinhebryear{#2}{\tmpf}%
8075     \ifnum #1 > 3
8076         \ifnum \tmpf=353
8077             \advance #3 by -1
8078         \fi
8079         \ifnum \tmpf=383
8080             \advance #3 by -1
8081         \fi

```



```

8082 \fi
8083 \ifnum #1 > 2
8084 \ifnum \tmpf=355
8085 \advance #3 by 1
8086 \fi
8087 \ifnum \tmpf=385
8088 \advance #3 by 1
8089 \fi
8090 \fi
8091 \global\bbl@cntcommon=#3\relax}%
8092 #3=\bbl@cntcommon}
8093 \def\bbl@absfromhebr#1#2#3#4{%
8094 {#4=#1\relax
8095 \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8096 \advance #4 by #1\relax
8097 \bbl@hebrrelapseddays{#3}{#1}%
8098 \advance #4 by #1\relax
8099 \advance #4 by -1373429
8100 \global\bbl@cntcommon=#4\relax}%
8101 #4=\bbl@cntcommon}
8102 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8103 {\countdef\tmpx= 17
8104 \countdef\tmpy= 18
8105 \countdef\tmpz= 19
8106 #6=#3\relax
8107 \global\advance #6 by 3761
8108 \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8109 \tmpz=1 \tmpy=1
8110 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8111 \ifnum \tmpx > #4\relax
8112 \global\advance #6 by -1
8113 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8114 \fi
8115 \advance #4 by -\tmpx
8116 \advance #4 by 1
8117 #5=#4\relax
8118 \divide #5 by 30
8119 \loop
8120 \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8121 \ifnum \tmpx < #4\relax
8122 \advance #5 by 1
8123 \tmpy=\tmpx
8124 \repeat
8125 \global\advance #5 by -1
8126 \global\advance #4 by -\tmpy}}
8127 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
8128 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8129 \def\bbl@ca@hebrew#1-#2-#3\@#4#5#6{%
8130 \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8131 \bbl@hebrfromgreg
8132 {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8133 {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
8134 \edef#4{\the\bbl@hebryear}%
8135 \edef#5{\the\bbl@hebrmonth}%
8136 \edef#6{\the\bbl@hebrday}}
8137 /ca-hebrew)

```

12.3 Persian

There is an algorithm written in TeX by Jabri, Abolhassani, Pournader and Esfahbod, created for the first versions of the FarsiTeX system (no longer available), but the original license is GPL, so its use with LPPL 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).

```

8138 (*ca-persian)
8139 \ExplSyntaxOn
8140 <<Compute Julian day>>
8141 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8142   2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8143 \def\bbl@ca@persian#1-#2-#3\@#4#5#6{%
8144   \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8145   \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8146     \bbl@afterfi\expandafter@gobble
8147   \fi\fi
8148   {\bbl@error{Year~out~of~range}{The~allowed~range~is~2013-2050}}}%
8149   \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8150   \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8151   \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8152   \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8153   \ifnum\bbl@tempc<\bbl@tempb
8154     \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8155     \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8156     \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8157     \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8158   \fi
8159   \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8160   \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8161   \edef#5{\fp_eval:n{% set Jalali month
8162     (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8163   \edef#6{\fp_eval:n{% set Jalali day
8164     (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6))}}}}
8165 \ExplSyntaxOff
8166 /ca-persian)

```

12.4 Coptic and Ethiopic

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

```

8167 (*ca-coptic)
8168 \ExplSyntaxOn
8169 <<Compute Julian day>>
8170 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
8171   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8172   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8173   \edef#4{\fp_eval:n{%
8174     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8175   \edef\bbl@tempc{\fp_eval:n{%
8176     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8177   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8178   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}}
8179 \ExplSyntaxOff
8180 /ca-coptic)
8181 (*ca-ethiopic)
8182 \ExplSyntaxOn
8183 <<Compute Julian day>>
8184 \def\bbl@ca@ethiopic#1-#2-#3\@#4#5#6{%
8185   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8186   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8187   \edef#4{\fp_eval:n{%
8188     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8189   \edef\bbl@tempc{\fp_eval:n{%
8190     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8191   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8192   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}}
8193 \ExplSyntaxOff

```

```
8194 </ca-ethiopic>
```

12.5 Buddhist

That's very simple.

```
8195 <*ca-buddhist>
8196 \def\bbl@ca@buddhist#1-#2-#3\@#4#5#6{%
8197   \edef#4{\number\numexpr#1+543\relax}%
8198   \edef#5{#2}%
8199   \edef#6{#3}}
8200 </ca-buddhist>
```

13 Support for Plain T_EX (plain.def)

13.1 Not renaming hyphen.tex

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

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

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

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

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

```
8201 <*bplain | blplain>
8202 \catcode`\{=1 % left brace is begin-group character
8203 \catcode`\}=2 % right brace is end-group character
8204 \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).

```
8205 \openin 0 hyphen.cfg
8206 \ifeof0
8207 \else
8208   \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.

```
8209 \def\input #1 {%
8210   \let\input\a
8211   \a hyphen.cfg
8212   \let\a\undefined
8213 }
8214 \fi
8215 </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`.

```
8216 <bplain>\a plain.tex
8217 <blplain>\a lplain.tex
```

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

```
8218 <bplain>\def\fmtname{babel-plain}
8219 <blplain>\def\fmtname{babel-lplain}
```

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

13.2 Emulating some \LaTeX features

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

```
8220 <<*Emulate LaTeX>> ≡
8221 \def\@empty{}
8222 \def\loadlocalcfg#1{%
8223   \openin0#1.cfg
8224   \ifeof0
8225     \closein0
8226   \else
8227     \closein0
8228     {immediate\write16{*****}%
8229     \immediate\write16{* Local config file #1.cfg used}%
8230     \immediate\write16{*}%
8231     }
8232   \input #1.cfg\relax
8233 \fi
8234 \@endofldf}
```

13.3 General tools

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

```
8235 \long\def\@firstofone#1{#1}
8236 \long\def\@firstoftwo#1#2{#1}
8237 \long\def\@secondoftwo#1#2{#2}
8238 \def\@nnil{\@nil}
8239 \def\@gobbletwo#1#2{}
8240 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
8241 \def\@star@or@long#1{%
8242   \@ifstar
8243   {\let\l@ngrel@x\relax#1}%
8244   {\let\l@ngrel@x\long#1}}
8245 \let\l@ngrel@x\relax
8246 \def\@car#1#2\@nil{#1}
8247 \def\@cdr#1#2\@nil{#2}
8248 \let\@typeset@protect\relax
8249 \let\protected@edef\edef
8250 \long\def\@gobble#1{}
8251 \edef\@backslashchar{\expandafter\@gobble\string\}
8252 \def\strip@prefix#1>{}
8253 \def\g@addto@macro#1#2{%
8254   \toks@\expandafter{#1#2}%
8255   \xdef#1{\the\toks@}}
8256 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8257 \def\@nameuse#1{\csname #1\endcsname}
8258 \def\@ifundefined#1{%
8259   \expandafter\ifx\csname#1\endcsname\relax
8260     \expandafter\@firstoftwo
8261   \else
8262     \expandafter\@secondoftwo
8263   \fi}
8264 \def\@expandtwoargs#1#2#3{%
8265   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8266 \def\zap@space#1 #2{%
8267   #1%
```

```

8268 \ifx#2\@empty\else\expandafter\zap@space\fi
8269 #2}
8270 \let\bbl@trace\@gobble
8271 \def\bbl@error#1#2{%
8272 \begingroup
8273 \newlinechar=`^^J
8274 \def\{^^J(babel) }%
8275 \errhelp{#2}\errmessage{\#1}%
8276 \endgroup}
8277 \def\bbl@warning#1{%
8278 \begingroup
8279 \newlinechar=`^^J
8280 \def\{^^J(babel) }%
8281 \message{\#1}%
8282 \endgroup}
8283 \let\bbl@infowarn\bbl@warning
8284 \def\bbl@info#1{%
8285 \begingroup
8286 \newlinechar=`^^J
8287 \def\{^^J}%
8288 \wlog{#1}%
8289 \endgroup}

```

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

```

8290 \ifx\@preamblecmds\undefined
8291 \def\@preamblecmds{}
8292 \fi
8293 \def\@onlypreamble#1{%
8294 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8295 \@preamblecmds\do#1}}
8296 \@onlypreamble\@onlypreamble

```

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

```

8297 \def\begin{document}{%
8298 \@begin{document}hook
8299 \global\let\@begin{document}hook\@undefined
8300 \def\do##1{\global\let##1\@undefined}%
8301 \@preamblecmds
8302 \global\let\do\noexpand}

8303 \ifx\@begin{document}hook\@undefined
8304 \def\@begin{document}hook{}
8305 \fi
8306 \@onlypreamble\@begin{document}hook
8307 \def\AtBeginDocument{\g@addto{macro}\@begin{document}hook}

```

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

```

8308 \def\AtEndOfPackage#1{\g@addto{macro}\@endoflfd{#1}}
8309 \@onlypreamble\AtEndOfPackage
8310 \def\@endoflfd{}
8311 \@onlypreamble\@endoflfd
8312 \let\bbl@afterlang\@empty
8313 \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.

```

8314 \catcode`\&=\z@
8315 \ifx&\if@files\@undefined
8316 \expandafter\let\csname if@files\expandafter\end{csname
8317 \csname iffalse\end{csname
8318 \fi
8319 \catcode`\&=4

```

Mimick \LaTeX 's commands to define control sequences.

```

8320 \def\newcommand{\@star@or@long\new@command}
8321 \def\new@command#1{%
8322   \@testopt{\@newcommand#1}0}
8323 \def\@newcommand#1[#2]{%
8324   \@ifnextchar [{\@xargdef#1[#2]}%
8325               {\@argdef#1[#2]}}
8326 \long\def\@argdef#1[#2]#3{%
8327   \@yargdef#1\@ne{#2}{#3}}
8328 \long\def\@xargdef#1[#2][#3]#4{%
8329   \expandafter\def\expandafter#1\expandafter{%
8330     \expandafter\@protected@testopt\expandafter #1%
8331     \csname\string#1\expandafter\endcsname{#3}}}%
8332 \expandafter\@yargdef \csname\string#1\endcsname
8333 \tw@{#2}{#4}}
8334 \long\def\@yargdef#1#2#3{%
8335   \@tempcnta#3\relax
8336   \advance \@tempcnta \@ne
8337   \let\@hash@\relax
8338   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
8339   \@tempcntb #2%
8340   \@whilenum\@tempcntb <\@tempcnta
8341   \do{%
8342     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
8343     \advance\@tempcntb \@ne}%
8344   \let\@hash@##%
8345   \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
8346 \def\providecommand{\@star@or@long\provide@command}
8347 \def\provide@command#1{%
8348   \begingroup
8349     \escapechar\m@ne\xdef\@gtempa{\string#1}%
8350   \endgroup
8351   \expandafter\@ifundefined\@gtempa
8352     {\def\reserved@a{\new@command#1}}%
8353     {\let\reserved@a\relax
8354      \def\reserved@a{\new@command\reserved@a}}%
8355   \reserved@a}%
8356 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
8357 \def\declare@robustcommand#1{%
8358   \edef\reserved@a{\string#1}%
8359   \def\reserved@b{#1}%
8360   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
8361   \edef#1{%
8362     \ifx\reserved@a\reserved@b
8363       \noexpand\x@protect
8364       \noexpand#1%
8365     \fi
8366     \noexpand\protect
8367     \expandafter\noexpand\csname
8368       \expandafter\@gobble\string#1 \endcsname
8369   }%
8370   \expandafter\new@command\csname
8371     \expandafter\@gobble\string#1 \endcsname
8372 }
8373 \def\x@protect#1{%
8374   \ifx\protect\@typeset@protect\else
8375     \@x@protect#1%
8376   \fi
8377 }
8378 \catcode\&=\z@ % Trick to hide conditionals
8379 \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`.

```
8380 \def\bbl@tempa{\csname newif\endcsname&ifin@}
8381 \catcode`\&=4
8382 \ifx\in@\@undefined
8383 \def\in@#1#2{%
8384   \def\in@@##1##2##3\in@{%
8385     \ifx\in@@#2\in@false\else\in@true\fi}%
8386   \in@@#2#1\in@\in@@}
8387 \else
8388 \let\bbl@tempa\@empty
8389 \fi
8390 \bbl@tempa
```

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

```
8391 \def\@ifpackagewith#1#2#3#4{#3}
```

The \TeX 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.

```
8392 \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 $\TeX 2_{\epsilon}$ versions; just enough to make things work in plain \TeX environments.

```
8393 \ifx\@tempcnta\@undefined
8394 \csname newcount\endcsname\@tempcnta\relax
8395 \fi
8396 \ifx\@tempcntb\@undefined
8397 \csname newcount\endcsname\@tempcntb\relax
8398 \fi
```

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

```
8399 \ifx\bye\@undefined
8400 \advance\count10 by -2\relax
8401 \fi
8402 \ifx\@ifnextchar\@undefined
8403 \def\@ifnextchar#1#2#3{%
8404   \let\reserved@d=#1%
8405   \def\reserved@a{#2}\def\reserved@b{#3}%
8406   \futurelet\@let@token\@ifnch}
8407 \def\@ifnch{%
8408   \ifx\@let@token\@sptoken
8409     \let\reserved@c\@ifnch
8410   \else
8411     \ifx\@let@token\reserved@d
8412       \let\reserved@c\reserved@a
8413     \else
8414       \let\reserved@c\reserved@b
8415     \fi
8416   \fi
8417   \reserved@c}
8418 \def\:{\let\@sptoken= }\: % this makes \@sptoken a space token
8419 \def\:{\@ifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
8420 \fi
8421 \def\@testopt#1#2{%
8422   \@ifnextchar[#{1}{#1[#{2}]}}
8423 \def\@protected@testopt#1{%
8424   \ifx\protect\@typeset@protect
```

```

8425 \expandafter\@testopt
8426 \else
8427 \@@protect#1%
8428 \fi}
8429 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8430 #2\relax}\fi}
8431 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8432 \else\expandafter\@gobble\fi{#1}}

```

13.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain $\text{T}_{\text{E}}\text{X}$ environment.

```

8433 \def\DeclareTextCommand{%
8434 \@@dec@text@cmd\providecommand
8435 }
8436 \def\ProvideTextCommand{%
8437 \@@dec@text@cmd\providecommand
8438 }
8439 \def\DeclareTextSymbol#1#2#3{%
8440 \@@dec@text@cmd\chardef#1{#2}#3\relax
8441 }
8442 \def\@dec@text@cmd#1#2#3{%
8443 \expandafter\def\expandafter#2%
8444 \expandafter{%
8445 \csname#3-cmd\expandafter\endcsname
8446 \expandafter#2%
8447 \csname#3\string#2\endcsname
8448 }%
8449 % \let\@ifdefinable\@rc@ifdefinable
8450 \expandafter#1\csname#3\string#2\endcsname
8451 }
8452 \def\@current@cmd#1{%
8453 \ifx\protect\@typeset@protect\else
8454 \noexpand#1\expandafter\@gobble
8455 \fi
8456 }
8457 \def\@changed@cmd#1#2{%
8458 \ifx\protect\@typeset@protect
8459 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8460 \expandafter\ifx\csname ?\string#1\endcsname\relax
8461 \expandafter\def\csname ?\string#1\endcsname{%
8462 \@changed@x@err{#1}%
8463 }%
8464 \fi
8465 \global\expandafter\let
8466 \csname\cf@encoding\string#1\expandafter\endcsname
8467 \csname ?\string#1\endcsname
8468 \fi
8469 \csname\cf@encoding\string#1%
8470 \expandafter\endcsname
8471 \else
8472 \noexpand#1%
8473 \fi
8474 }
8475 \def\@changed@x@err#1{%
8476 \errhelp{Your command will be ignored, type <return> to proceed}%
8477 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8478 \def\DeclareTextCommandDefault#1{%
8479 \DeclareTextCommand#1?%
8480 }
8481 \def\ProvideTextCommandDefault#1{%
8482 \ProvideTextCommand#1?%
8483 }

```



```

8484 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8485 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8486 \def\DeclareTextAccent#1#2#3{%
8487   \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
8488 }
8489 \def\DeclareTextCompositeCommand#1#2#3#4{%
8490   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8491   \edef\reserved@b{\string##1}%
8492   \edef\reserved@c{%
8493     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8494   \ifx\reserved@b\reserved@c
8495     \expandafter\expandafter\expandafter\ifx
8496       \expandafter\@car\reserved@a\relax\relax\@nil
8497       \@text@composite
8498   \else
8499     \edef\reserved@b##1{%
8500       \def\expandafter\noexpand
8501         \csname#2\string#1\endcsname###1{%
8502           \noexpand\@text@composite
8503             \expandafter\noexpand\csname#2\string#1\endcsname
8504             ###1\noexpand\@empty\noexpand\@text@composite
8505             {##1}%
8506         }%
8507     }%
8508     \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8509   \fi
8510   \expandafter\def\csname\expandafter\string\csname
8511     #2\endcsname\string#1-\string#3\endcsname{#4}
8512 \else
8513   \errhelp{Your command will be ignored, type <return> to proceed}%
8514   \errmessage{\string\DeclareTextCompositeCommand\space used on
8515     inappropriate command \protect#1}
8516 \fi
8517 }
8518 \def\@text@composite#1#2#3\@text@composite{%
8519   \expandafter\@text@composite@x
8520     \csname\string#1-\string#2\endcsname
8521 }
8522 \def\@text@composite@x#1#2{%
8523   \ifx#1\relax
8524     #2%
8525   \else
8526     #1%
8527   \fi
8528 }
8529 %
8530 \def\@strip@args#1:#2-#3\@strip@args{#2}
8531 \def\DeclareTextComposite#1#2#3#4{%
8532   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8533   \bgroup
8534     \lccode`\@=#4%
8535     \lowercase{%
8536       \egroup
8537       \reserved@a @%
8538     }%
8539 }
8540 %
8541 \def\UseTextSymbol#1#2{#2}
8542 \def\UseTextAccent#1#2#3{}
8543 \def\@use@text@encoding#1{}
8544 \def\DeclareTextSymbolDefault#1#2{%
8545   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8546 }

```

```

8547 \def\DeclareTextAccentDefault#1#2{%
8548   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8549 }
8550 \def\cf@encoding{OT1}

```

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

```

8551 \DeclareTextAccent{"}{OT1}{127}
8552 \DeclareTextAccent{'}{OT1}{19}
8553 \DeclareTextAccent{^}{OT1}{94}
8554 \DeclareTextAccent{\`}{OT1}{18}
8555 \DeclareTextAccent{~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for PLAIN $\text{\texttt{\textbackslash T E X}}$.

```

8556 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8557 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
8558 \DeclareTextSymbol{\textquoteleft}{OT1}{`\'}
8559 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
8560 \DeclareTextSymbol{\i}{OT1}{16}
8561 \DeclareTextSymbol{\ss}{OT1}{25}

```

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

```

8562 \ifx\scriptsize\@undefined
8563   \let\scriptsize\sevenrm
8564 \fi

```

And a few more “dummy” definitions.

```

8565 \def\language{english}%
8566 \let\bbl@opt@shorthands\@nnil
8567 \def\bbl@ifshorthand#1#2#3{#2}%
8568 \let\bbl@language@opts\@empty
8569 \let\bbl@ensureinfo\@gobble
8570 \let\bbl@provide@locale\relax
8571 \ifx\babeloptionstrings\@undefined
8572   \let\bbl@opt@strings\@nnil
8573 \else
8574   \let\bbl@opt@strings\babeloptionstrings
8575 \fi
8576 \def\BabelStringsDefault{generic}
8577 \def\bbl@tempa{normal}
8578 \ifx\babeloptionmath\bbl@tempa
8579   \def\bbl@mathnormal{\noexpand\textormath}
8580 \fi
8581 \def\AfterBabelLanguage#1#2{}
8582 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
8583 \let\bbl@afterlang\relax
8584 \def\bbl@opt@safe{BR}
8585 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
8586 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
8587 \expandafter\newif\csname ifbbl@single\endcsname
8588 \chardef\bbl@bidimode\z@
8589 <</Emulate LaTeX>>

```

A proxy file:

```

8590 <*\plain>
8591 \input babel.def
8592 </\plain>

```

14 Acknowledgements

I would like to thank all who volunteered as β -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the

documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful. There are also many contributors for specific languages, which are mentioned in the respective files. Without them, babel just wouldn't exist.

References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national \LaTeX styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The \TeX book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport, *\LaTeX , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in: \TeX hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018
- [10] Hubert Partl, *German \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).