

Babel

Code

Version 24.1
2024/01/07

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Localization and
internationalization

Unicode

TeX

pdfTeX

LuaTeX

XeTeX

Contents

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

| | | |
|-----------|---|------------|
| 10 | Support for interchar | 103 |
| 10.1 | Layout | 105 |
| 10.2 | 8-bit TeX | 107 |
| 10.3 | LuaTeX | 108 |
| 10.4 | Southeast Asian scripts | 114 |
| 10.5 | CJK line breaking | 115 |
| 10.6 | Arabic justification | 117 |
| 10.7 | Common stuff | 122 |
| 10.8 | Automatic fonts and ids switching | 122 |
| 10.9 | Bidi | 128 |
| 10.10 | Layout | 130 |
| 10.11 | Lua: transforms | 137 |
| 10.12 | Lua: Auto bidi with <code>basic</code> and <code>basic-r</code> | 146 |
| 11 | Data for CJK | 157 |
| 12 | The ‘nil’ language | 157 |
| 13 | Calendars | 158 |
| 13.1 | Islamic | 158 |
| 13.2 | Hebrew | 160 |
| 13.3 | Persian | 164 |
| 13.4 | Coptic and Ethiopic | 164 |
| 13.5 | Buddhist | 165 |
| 14 | Support for Plain TeX (<code>plain.def</code>) | 166 |
| 14.1 | Not renaming <code>hyphen.tex</code> | 166 |
| 14.2 | Emulating some \LaTeX features | 167 |
| 14.3 | General tools | 167 |
| 14.4 | Encoding related macros | 171 |
| 15 | Acknowledgements | 174 |

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

1 Identification and loading of required files

Code documentation is still under revision.

The babel package after unpacking consists of the following files:

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

babel.def is loaded by Plain.

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

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

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

There some additional tex, def and lua files

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

2 locale directory

A required component of babel is a set of ini files with basic definitions for about 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=24.1>>
2 <<date=2024/01/07>>
```

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

```

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

```

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

```

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

```

Two further tools. \bbl@ifsamestring first expand its arguments and then compare their expansion (sanitized, so that the catcodes do not matter). \bbl@engine takes the following values: 0 is 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.ldf-h@@k\endcsname}%

```

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

```

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

```

3.3 base

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

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

```

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

```

3.4 key=value options and other general option

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

```
292 \bbl@trace{key=value and another general options}
293 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
294 \def\bbl@tempb#1.#2{% Remove trailing dot
295   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
296 \def\bbl@tempe#1=#2\@@{%
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@bcptoptions]{\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 \LaTeX 2_ϵ . 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 \LaTeX 2_ϵ , 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@filesw
1270     \immediate\write\@mainaux{\catcode`\string#2\active}%
1271   \fi}%
1272 \expandafter\bbl@add@special\csname#2\endcsname
1273 \catcode`#2\active
1274 \fi

```

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

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

```

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

```

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

`\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 `\ifincsname` is available. If there is, the expansion will be more robust.

```

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

```

```

1359     \fi}}
1360 \endgroup

```

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

```

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

```

`\@notshorthand`

```

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

```

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

```

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

```

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

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

```

1481 \def\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@es` One of the internal macros that are involved in substituting `\prime` for each right quote in `\bbl@pr@mes` mathmode is `\prim@es`. 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@es{%
1539   \prime\futurelet\@let@token\bbl@pr@mes}
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@mes{%
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 selected using the `\f@encoding` macro. Therefore we define two macros here to store the position of the character in these encodings.

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

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

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

4.6 Language attributes

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

1611 \def\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\scode`\.=\@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{\scode`##1=\the\scode`##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\scode`##1=##2\relax
1673 \babel@savevariable{\scode`##1}%
1674 \scode`##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\scode`##1=##3\relax
1680 \babel@savevariable{\scode`##1}%
1681 \scode`##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 following option is currently no-op. It was meant for the deprecated `\SetCase`.

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

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

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

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

```

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

```

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

```

```

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

```

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

```

```

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

```

Now we define commands to be used inside \StartBabelCommands.

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

The following package options control the behavior of hyphenation mapping.

```

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

```


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

```

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

```

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

```

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

```

```

2036 \fi}
2037 % \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.

```

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

```

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

```

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

```

4.12 Making glyphs available

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

4.12.1 Quotation marks

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

4.12.2 Letters

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

```

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

```

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

```

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

```

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

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

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

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

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

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

```
2143 \DeclareTextCommand{\SS}{OT1}{SS}
2144 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

4.12.3 Shorthands for quotation marks

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

`\glq` The ‘german’ single quotes.

```
\grq
2145 \ProvideTextCommandDefault{\glq}{%
2146   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

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

`\glqq` The ‘german’ double quotes.

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

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

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

```

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

```

\flq The ‘french’ single guillemets.

```

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

```

\flqq The ‘french’ double guillemets.

```

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

```

4.12.4 Umlauts and tremas

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

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

```

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

```

\lower@umlaut The command \lower@umlaut is used to position the “\” closer to the letter.

We want the umlaut character lowered, nearer to the letter. To do this we need an extra *⟨dimen⟩* register.

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

4.13 Layout

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

```

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

```

```

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

```

4.14 Load engine specific macros

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

```

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

```

4.15 Creating and modifying languages

Continue with \LaTeX only.

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

```

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

```

```

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

```



```

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

```

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

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

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

```

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

```

```

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

```

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.)

```

2725 \def\bbl@load@basic#1{%
2726 \ifcase\bbl@howloaded\or\or

```

```

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

```

The hyphenrules option is handled with an auxiliary macro. This macro is called in three cases: when a language is first declared with \babelprovide, with hyphenrules and with import.

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```



```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

3061 \expandafter\bbbl@buildifcase\bbbl@tempb* \ \ % Space after \ \
3062 \bbbl@csarg{\global\expandafter\let}{\cnt@#1@\language\name}\bbbl@tempa
3063 \fi}

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

Dates will require some macros for the basic formatting. They may be redefined by language, so “semi-public” names (camel case) are used. Oddly enough, the CLDR places particles like “de” inconsistently in either in the date or in the month name. Note after `\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).

```

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

```



```

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

```

Transforms.

```

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

```

```

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

```

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

```

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

```

```

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

```

The following ini reader ignores everything but the identification section. It is called when a font is defined (ie, when the language is first selected) to know which script/language must be enabled. This means we must make sure a few characters are not active. The ini is not read directly, but with a proxy tex file named as the language (which means any code in it must be skipped, too).

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

5 Adjusting the Babel bahavior

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

```

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

```

```

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

```

```

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

```

5.1 Cross referencing macros

The \LaTeX book states:

The *key* argument is any sequence of letters, digits, and punctuation symbols; upper- and lowercase letters are regarded as different.

When the above quote should still be true when a document is typeset in a language that has active characters, special care has to be taken of the category codes of these characters when they appear in an argument of the cross referencing macros.

When a cross referencing command processes its argument, all tokens in this argument should be character tokens with category ‘letter’ or ‘other’.

The following package options control which macros are to be redefined.

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

`\@newl@bel` First we open a new group to keep the changed setting of `\protect` local and then we set the `@safe@actives` switch to true to make sure that any shorthand that appears in any of the arguments immediately expands to its non-active self.

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

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

```
3770 \CheckCommand*\@testdef[3]{%
3771   \def\reserved@a{#3}%
3772   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3773   \else
3774     \@tempswatrue
3775   \fi}
```

Now that we made sure that `\@testdef` still has the same definition we can rewrite it. First we make the shorthands ‘safe’. Then we use `\bbl@tempa` as an ‘alias’ for the macro that contains the label which is being checked. Then we define `\bbl@tempb` just as `\@newl@bel` does it. When the label is defined we replace the definition of `\bbl@tempa` by its meaning. If the label didn’t change, `\bbl@tempa` and `\bbl@tempb` should be identical macros.

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

```

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

```

`\@citex` The macro used to cite from a bibliography, `\cite`, uses an internal macro, `\@citex`. It is this internal macro that picks up the argument(s), so we redefine this internal macro and leave `\cite` alone. The first argument is used for typesetting, so the shorthands need only be deactivated in the second argument.

```

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

```

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

```

3820 \AtBeginDocument{%
3821 \@ifpackageloaded{natbib}{%

```

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

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

```

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

```

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

```

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

```

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

```

3831 \bbl@redefine\nocite#1{%
3832 \@safe@activestruel\org@nocite{#1}\@safe@activesfalse}

```

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

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

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

```
3836 \def\bbl@bibcite#1#2{%
3837   \org@bibcite{#1}{\@safe@activesfalse#2}}
```

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

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

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

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

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

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

5.2 Marks

`\markright` Because the output routine is asynchronous, we must pass the current language attribute to the head lines. To achieve this we need to adapt the definition of `\markright` and `\markboth` somewhat. However, headlines and footlines can contain text outside marks; for that we must take some actions in the output routine if the 'headfoot' options is used.

We need to make some redefinitions to the output routine to avoid an endless loop and to correctly handle the page number in bidi documents.

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

```

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

```

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

```

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

```

5.3 Preventing clashes with other packages

5.3.1 `ifthen`

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

```

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

```

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

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

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

```

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

```

```

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

```

5.3.2 varioref

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

```

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

```

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

```

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

```

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

```

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

```

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

```

5.4 Encoding and fonts

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

`\ensureascii`

```

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

```

```
3994 \fi}
```

Now comes the old deprecated stuff (with a little change in 3.9l, for fontspec). The first thing we need to do is to determine, at `\begin{document}`, which latin fontencoding to use.

`\latinencoding` When text is being typeset in an encoding other than ‘latin’ (OT1 or T1), it would be nice to still have Roman numerals come out in the Latin encoding. So we first assume that the current encoding at the end of processing the package is the Latin encoding.

```
3995 \AtEndOfPackage{\edef\latinencoding{\cf@encoding}}
```

But this might be overruled with a later loading of the package `fontenc`. Therefore we check at the execution of `\begin{document}` whether it was loaded with the T1 option. The normal way to do this (using `\ifpackageloaded`) is disabled for this package. Now we have to revert to parsing the internal macro `\@filelist` which contains all the filenames loaded.

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

`\latintext` Then we can define the command `\latintext` which is a declarative switch to a latin font-encoding. Usage of this macro is deprecated.

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

`\textlatin` This command takes an argument which is then typeset using the requested font encoding. In order to avoid many encoding switches it operates in a local scope.

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

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

```
4024 \def\bb@patchfont#1{\AddToHook{selectfont}{#1}}
```

5.5 Basic bidi support

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

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

```

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

```


4075 \fi

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

```

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

```

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

```

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

```

```

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

```

The following command is executed only if there is a right-to-left script (once). It activates the `\everypar` hack for xetex, to properly handle the par direction. Note text and par dirs are decoupled to some extent (although not completely).

```

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

```

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

```

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

```

```

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

```

5.6 Local Language Configuration

`\loadlocalcfg` At some sites it may be necessary to add site-specific actions to a language definition file. This can be done by creating a file with the same name as the language definition file, but with the extension `.cfg`. For instance the file `norsk.cfg` will be loaded when the language definition file `norsk.ldf` is loaded.

For plain-based formats we don't want to override the definition of `\loadlocalcfg` from `plain.def`.

```

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

```

5.7 Language options

Languages are loaded when processing the corresponding option *except* if a main language has been set. In such a case, it is not loaded until all options has been processed. The following macro inputs the `ldf` file and does some additional checks (`\input` works, too, but possible errors are not caught).

```

4209 \bbl@trace{Language options}
4210 \let\bbl@afterlang\relax
4211 \let\BabelModifiers\relax
4212 \let\bbl@loaded\@empty
4213 \def\bbl@load@language#1{%
4214   \InputIfFileExists{#1.ldf}%
4215   {\edef\bbl@loaded{\CurrentOption
4216     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4217     \expandafter\let\expandafter\bbl@afterlang
4218     \csname\CurrentOption.ldf-h@k\endcsname
4219     \expandafter\let\expandafter\BabelModifiers
4220     \csname bbl@mod@\CurrentOption\endcsname
4221     \bbl@exp{\AtBeginDocument{%
4222       \bbl@usehooks@lang{\CurrentOption}{begindocument}{\CurrentOption}}}%
4223     {\IfFileExists{babel-#1.tex}%
4224       {\def\bbl@tempa{%
4225         .\There is a locale ini file for this language.\\%
4226         If it's the main language, try adding `provide='\\%
4227         to the babel package options}}%
4228       {\let\bbl@tempa\empty}%
4229       \bbl@error{%
4230         Unknown option '\CurrentOption'. Either you misspelled it\\%
4231         or the language definition file \CurrentOption.ldf\\%
4232         was not found%
4233         \bbl@tempa}%
4234       Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4235       activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4236       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.

```

4237 \def\bbl@try@load@lang#1#2#3{%
4238   \IfFileExists{\CurrentOption.ldf}%
4239     {\bbl@load@language{\CurrentOption}}%
4240     {\#1\bbl@load@language{\#2}\#3}}
4241 %
4242 \DeclareOption{hebrew}{%
4243   \input{rlbabel.def}%
4244   \bbl@load@language{hebrew}}
4245 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4246 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4247 \DeclareOption{northensami}{\bbl@try@load@lang{}{samin}{}}
4248 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4249 \DeclareOption{polutonikogreek}{%
4250   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4251 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4252 \DeclareOption{scottishgaelic}{\bbl@try@load@lang{}{scottish}{}}
4253 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4254 \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.

```

4255 \ifx\bbl@opt@config\@nnil
4256   \ifpackagewith{babel}{noconfigs}{}%
4257     {\InputIfFileExists{bblopts.cfg}%
4258       {\typeout{*****^J%
4259                * Local config file bblopts.cfg used^^J%
4260                *}}}%
4261     {}}%
4262 \else
4263   \InputIfFileExists{\bbl@opt@config.cfg}%
4264     {\typeout{*****^J%
4265                * Local config file \bbl@opt@config.cfg used^^J%
4266                *}}}%
4267     {\bbl@error{%
4268       Local config file '\bbl@opt@config.cfg' not found}{%
4269       Perhaps you misspelled it.}}}%
4270 \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 `babel@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.

```

4271 \ifx\bbl@opt@main\@nnil
4272   \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4273     \let\bbl@tempb\@empty
4274     \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4275     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{\#1,\bbl@tempb}}%
4276     \bbl@foreach\bbl@tempb{%   \bbl@tempb is a reversed list
4277       \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4278         \ifodd\bbl@iniflag % = * =
4279           \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
4280         \else % n +=
4281           \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4282       \fi
4283     \fi}%
4284 \fi
4285 \else
4286   \bbl@info{Main language set with 'main='. Except if you have\\%
4287     problems, prefer the default mechanism for setting\\%

```

```

4288         the main language, ie, as the last declared.\\%
4289         Reported}
4290 \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).

```

4291 \ifx\bbbl@opt@main\@nnil\else
4292   \bbbl@ncarg\let\bbbl@loadmain{ds@\bbbl@opt@main}%
4293   \expandafter\let\csname ds@\bbbl@opt@main\endcsname\relax
4294 \fi

```

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

```

4295 \bbbl@foreach\bbbl@language@opts{%
4296   \def\bbbl@tempa{#1}%
4297   \ifx\bbbl@tempa\bbbl@opt@main\else
4298     \ifnum\bbbl@iniflag<\tw@ % 0 0 (other = ldf)
4299       \bbbl@ifunset{ds@#1}%
4300       {\DeclareOption{#1}{\bbbl@load@language{#1}}}%
4301       {}%
4302     \else % + * (other = ini)
4303       \DeclareOption{#1}{%
4304         \bbbl@ldfinit
4305         \babelprovide[import]{#1}%
4306         \bbbl@afterldf{}}%
4307     \fi
4308   \fi}
4309 \bbbl@foreach\@classoptionslist{%
4310   \def\bbbl@tempa{#1}%
4311   \ifx\bbbl@tempa\bbbl@opt@main\else
4312     \ifnum\bbbl@iniflag<\tw@ % 0 0 (other = ldf)
4313       \bbbl@ifunset{ds@#1}%
4314       {\IfFileExists{#1.ldf}%
4315        {\DeclareOption{#1}{\bbbl@load@language{#1}}}%
4316        {}}%
4317     \else % + * (other = ini)
4318       \IfFileExists{babel-#1.tex}%
4319       {\DeclareOption{#1}{%
4320         \bbbl@ldfinit
4321         \babelprovide[import]{#1}%
4322         \bbbl@afterldf{}}}%
4323       {}%
4324     \fi
4325   \fi}
4326 \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):

```

4327 \def\AfterBabelLanguage#1{%
4328   \bbbl@ifsamestring\CurrentOption{#1}{\global\bbbl@add\bbbl@afterlang{}}
4329 \DeclareOption*{}
4330 \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.

```

4331 \bbbl@trace{Option 'main'}
4332 \ifx\bbbl@opt@main\@nnil
4333   \edef\bbbl@tempa{\@classoptionslist,\bbbl@language@opts}

```

```

4334 \let\bbl@tempc\@empty
4335 \edef\bbl@templ{\bbl@loaded,}
4336 \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4337 \bbl@for\bbl@tempb\bbl@tempa{%
4338   \edef\bbl@tempd{\bbl@tempb,}%
4339   \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4340   \bbl@xin{\bbl@tempd}{\bbl@templ}%
4341   \ifin@{\edef\bbl@tempc{\bbl@tempb}\fi}
4342 \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4343 \expandafter\bbl@tempa\bbl@loaded,\@nnil
4344 \ifx\bbl@tempb\bbl@tempc\else
4345   \bbl@warning{%
4346     Last declared language option is '\bbl@tempc',\%
4347     but the last processed one was '\bbl@tempb'.\%
4348     The main language can't be set as both a global\%
4349     and a package option. Use 'main=\bbl@tempc' as\%
4350     option. Reported}
4351 \fi
4352 \else
4353 \ifodd\bbl@iniflag % case 1,3 (main is ini)
4354   \bbl@ldfinit
4355   \let\CurrentOption\bbl@opt@main
4356   \bbl@exp{% \bbl@opt@provide = empty if *
4357     \\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4358   \bbl@afterldf{}
4359   \DeclareOption{\bbl@opt@main}{}
4360 \else % case 0,2 (main is ldf)
4361   \ifx\bbl@loadmain\relax
4362     \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4363   \else
4364     \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4365   \fi
4366   \ExecuteOptions{\bbl@opt@main}
4367   \@namedef{ds@\bbl@opt@main}{}%
4368 \fi
4369 \DeclareOption*{}
4370 \ProcessOptions*
4371 \fi
4372 \bbl@exp{%
4373   \\AtBeginDocument{\\bbl@usehooks@lang{/}{begindocument}{}}}%
4374 \def\AfterBabelLanguage{%
4375   \bbl@error
4376   {Too late for \string\AfterBabelLanguage}%
4377   {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.

4378 \ifx\bbl@main@language\@undefined
4379   \bbl@info{%
4380     You haven't specified a language as a class or package\%
4381     option. I'll load 'nil'. Reported}
4382   \bbl@load@language{nil}
4383 \fi
4384 \end{package}

```

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

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

Because plain T_EX users might want to use some of the features of the babel system too, care has to be taken that plain T_EX can process the files. For this reason the current format will have to be checked

in a number of places. Some of the code below is common to plain $\text{T}_{\text{E}}\text{X}$ and $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$, some of it is for the $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ 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`

```
4385 <*kernel>
4386 \let\bbl@onlyswitch\@empty
4387 \input babel.def
4388 \let\bbl@onlyswitch\@undefined
4389 </kernel>
4390 <*patterns>
```

7 Loading hyphenation patterns

The following code is meant to be read by $\text{init}_{\text{E}}\text{X}$ because it should instruct $\text{T}_{\text{E}}\text{X}$ 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.

```
4391 <<Make sure ProvidesFile is defined>>
4392 \ProvidesFile{hyphen.cfg}[<<date>> v<<version>> Babel hyphens]
4393 \xdef\bbl@format{\jobname}
4394 \def\bbl@version{<<version>>}
4395 \def\bbl@date{<<date>>}
4396 \ifx\AtBeginDocument\@undefined
4397   \def\@empty{}
4398 \fi
4399 <<Define core switching macros>>
```

`\process@line` Each line in the file `language.dat` is processed by `\process@line` after it is read. The first thing this macro does is to check whether the line starts with `=`. When the first token of a line is an `=`, the macro `\process@synonym` is called; otherwise the macro `\process@language` will continue.

```
4400 \def\process@line#1#2 #3 #4 {%
4401   \ifx=#1%
4402     \process@synonym{#2}%
4403   \else
4404     \process@language{#1#2}{#3}{#4}%
4405   \fi
4406   \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.

```
4407 \toks@{}
4408 \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.

```
4409 \def\process@synonym#1{%
4410   \ifnum\last@language=\m@ne
4411     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4412   \else
4413     \expandafter\chardef\csname l@#1\endcsname\last@language
4414     \wlog{\string\l@#1=\string\language\the\last@language}%
4415     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4416       \csname\language\hyphenmins\endcsname
4417     \let\bbl@elt\relax
4418     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4419   \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.

```

4420 \def\process@language#1#2#3{%
4421   \expandafter\addlanguage\csname l@#1\endcsname
4422   \expandafter\language\csname l@#1\endcsname
4423   \edef\language#1{%
4424     \bbl@hook@everylanguage{#1}%
4425     % > luatex
4426     \bbl@get@enc#1:.\@@@
4427     \begingroup
4428       \lefthyphenmin\m@ne
4429       \bbl@hook@loadpatterns{#2}%
4430       % > luatex
4431       \ifnum\lefthyphenmin=\m@ne
4432       \else
4433         \expandafter\xdef\csname #1hyphenmins\endcsname{%
4434           \the\lefthyphenmin\the\righthyphenmin}%
4435       \fi
4436     \endgroup
4437     \def\bbl@tempa{#3}%
4438     \ifx\bbl@tempa\@empty\else
4439       \bbl@hook@loadexceptions{#3}%
4440       % > luatex
4441     \fi
4442     \let\bbl@elt\relax
4443     \edef\bbl@languages{%
4444       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4445     \ifnum\the\language=\z@
4446       \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4447         \set@hyphenmins\tw@\thr@@\relax
4448       \else
4449         \expandafter\expandafter\expandafter\set@hyphenmins
4450         \csname #1hyphenmins\endcsname
4451       \fi
4452       \the\toks@
4453       \toks@{}%
4454     \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.

```
4455 \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.

```
4456 \def\bbl@hook@everylanguage#1{}
4457 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4458 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4459 \def\bbl@hook@loadkernel#1{%
4460   \def\addlanguage{\csname newlanguage\endcsname}%
4461   \def\adddialect##1##2{%
4462     \global\chardef##1##2\relax
4463     \wlog{\string##1 = a dialect from \string\language##2}}%
4464   \def\iflanguage##1{%
4465     \expandafter\ifx\csname l@##1\endcsname\relax
4466       \@nolanerr{##1}%
4467     \else
4468       \ifnum\csname l@##1\endcsname=\language
4469         \expandafter\expandafter\expandafter\@firstoftwo
4470       \else
4471         \expandafter\expandafter\expandafter\@secondoftwo
4472       \fi
4473     \fi}%
4474   \def\providehyphenmins##1##2{%
4475     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4476       \@namedef{##1hyphenmins}{##2}%
4477     \fi}%
4478   \def\set@hyphenmins##1##2{%
4479     \leftthyphenmin##1\relax
4480     \rightthyphenmin##2\relax}%
4481   \def\selectlanguage{%
4482     \errhelp{Selecting a language requires a package supporting it}%
4483     \errmessage{Not loaded}}%
4484   \let\foreignlanguage\selectlanguage
4485   \let\otherlanguage\selectlanguage
4486   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4487   \def\bbl@usehooks##1##2{% TODO. Temporary!!
4488     \def\setlocale{%
4489       \errhelp{Find an armchair, sit down and wait}%
4490       \errmessage{Not yet available}}%
4491     \let\uselocale\setlocale
4492     \let\locale\setlocale
4493     \let\selectlocale\setlocale
4494     \let\localename\setlocale
4495     \let\textlocale\setlocale
4496     \let\textlanguage\setlocale
4497     \let\languagetext\setlocale}
4498 \begingroup
4499 \def\AddBabelHook#1#2{%
4500   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4501     \def\next{\toks1}%
4502   \else
4503     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4504   \fi
4505   \next}
4506 \ifx\directlua\@undefined
4507   \ifx\XeTeXinputencoding\@undefined\else
4508     \input xebabel.def
4509   \fi
4510 \else
4511   \input luababel.def
```

```

4512 \fi
4513 \openin1 = babel-\bbl@format.cfg
4514 \ifeof1
4515 \else
4516 \input babel-\bbl@format.cfg\relax
4517 \fi
4518 \closein1
4519 \endgroup
4520 \bbl@hook@loadkernel{switch.def}

```

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

```

4521 \openin1 = language.dat

```

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

```

4522 \def\language{english}%
4523 \ifeof1
4524 \message{I couldn't find the file language.dat,\space
4525         I will try the file hyphen.tex}
4526 \input hyphen.tex\relax
4527 \chardef\l@english\z@
4528 \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`.

```

4529 \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.

```

4530 \loop
4531 \endlinechar\m@ne
4532 \read1 to \bbl@line
4533 \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.

```

4534 \if T\ifeof1\fi T\relax
4535 \ifx\bbl@line\empty\else
4536 \edef\bbl@line{\bbl@line\space\space\space}%
4537 \expandafter\process@line\bbl@line\relax
4538 \fi
4539 \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.

```

4540 \begingroup
4541 \def\bbl@elt#1#2#3#4{%
4542 \global\language=#2\relax
4543 \gdef\language{#1}%
4544 \def\bbl@elt##1##2##3##4{}}%
4545 \bbl@languages
4546 \endgroup
4547 \fi
4548 \closein1

```

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

```

4549 \if/\the\toks@\else
4550 \errhelp{language.dat loads no language, only synonyms}
4551 \errmessage{Orphan language synonym}
4552 \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.

```

4553 \let\bbl@line\@undefined
4554 \let\process@line\@undefined
4555 \let\process@synonym\@undefined
4556 \let\process@language\@undefined
4557 \let\bbl@get@enc\@undefined
4558 \let\bbl@hyph@enc\@undefined
4559 \let\bbl@tempa\@undefined
4560 \let\bbl@hook@loadkernel\@undefined
4561 \let\bbl@hook@everylanguage\@undefined
4562 \let\bbl@hook@loadpatterns\@undefined
4563 \let\bbl@hook@loadexceptions\@undefined
4564 \patterns)

```

Here the code for `iniTeX` ends.

8 Font handling with fontspec

Add the bidi 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 bidi [misplaced].

```

4565 <<(*More package options)>> ≡
4566 \chardef\bbl@bidimode\z@
4567 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4568 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4569 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4570 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4571 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4572 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4573 <</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.

```

4574 <<(*Font selection)>> ≡
4575 \bbl@trace{Font handling with fontspec}
4576 \ifx\ExplSyntaxOn\@undefined\else
4577   \def\bbl@fs@warn@nx#1#2{% \bbl@tempfs is the original macro
4578     \in@{,#1,}{,no-script,language-not-exist,}%
4579     \ifin@else\bbl@tempfs@nx{#1}{#2}\fi}
4580   \def\bbl@fs@warn@nxx#1#2#3{%
4581     \in@{,#1,}{,no-script,language-not-exist,}%
4582     \ifin@else\bbl@tempfs@nxx{#1}{#2}{#3}\fi}
4583   \def\bbl@loadfontspec{%
4584     \let\bbl@loadfontspec\relax
4585     \ifx\fontspec\@undefined
4586       \usepackage{fontspec}%
4587     \fi}%
4588 \fi
4589 \@onlypreamble\babelfont
4590 \newcommand\babelfont[2][]{% 1=langs/scripts 2=fam
4591   \bbl@foreach{#1}{%
4592     \expandafter\ifx\csname date##1\endcsname\relax
4593       \IfFileExists{babel-##1.tex}%
4594       {\babelprovide{##1}}%
4595     }%
4596   \fi}%
4597 \edef\bbl@tempa{#1}%
4598 \def\bbl@tempb{#2}% Used by \bbl@bblfont

```

```

4599 \bbl@loadfontspec
4600 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4601 \bbl@bblfont}
4602 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4603 \bbl@ifunset{\bbl@tempb family}%
4604 {\bbl@providefam{\bbl@tempb}}%
4605 {}%
4606 % For the default font, just in case:
4607 \bbl@ifunset\bbl@lsys{\language}\bbl@provide@lsys{\language}}{}%
4608 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4609 {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
4610 \bbl@exp{%
4611 \let<\bbl@>\bbl@tempb dflt@\language>\<\bbl@>\bbl@tempb dflt@>%
4612 \\<\bbl@font@set<\bbl@>\bbl@tempb dflt@\language>%
4613 \<\bbl@tempb default>\<\bbl@tempb family>}}%
4614 {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4615 \bbl@csarg\def{\bbl@tempb dflt@#1}{<>{#1}{#2}}}}}%

```

```

4616 \def\bbl@providefam#1{%
4617   \bbl@exp{%
4618     \\newcommand\<#1default>{%} Just define it
4619     \\bbl@add@list\\bbl@font@fams{#1}%
4620     \\DeclareRobustCommand\<#1family>{%
4621       \\not@math@alphabet\<#1family>\relax
4622       % \\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4623       \\fontfamily\<#1default>%
4624       \<if>\\UseHooks\\@undefined\<else>\\UseHook{#1family}\<fi>%
4625       \\selectfont}%
4626       \\DeclareTextFontCommand{\<text#1>}\<#1family>}}}
```

```

4627 \def\bbbl@nostdfont#1{%
4628   \bbbl@ifunset{\bbbl@WFF@\f@family}%
4629     {\bbbl@csarg\gdef{\WFF@\f@family}{}}% Flag, to avoid dupl warns
4630     \bbbl@infowarn{The current font is not a babel standard family:\%
4631       #1%
4632       \fontname\font\\%
4633       There is nothing intrinsically wrong with this warning, and\\%
4634       you can ignore it altogether if you do not need these\\%
4635       families. But if they are used in the document, you should be\\%
4636       aware 'babel' will not set Script and Language for them, so\\%
4637       you may consider defining a new family with \string\babelfont.\\%
4638       See the manual for further details about \string\babelfont.\\%
4639       Reported}}
4640   {}}%
4641 \gdef\bbbl@switchfont{%
4642   \bbbl@ifunset{\bbbl@lsys@\language name}{\bbbl@provide@lsys{\language name}}{}}%
4643   \bbbl@exp{% eg Arabic -> arabic
4644     \lowercase{\edef\\bbbl@tempa{\bbbl@cl{iname}}}}%
4645   \bbbl@foreach\bbbl@font@fams{%
4646     \bbbl@ifunset{\bbbl@##1dflt@\language name}% (1) language?
4647       {\bbbl@ifunset{\bbbl@##1dflt@*\bbbl@tempa}% (2) from script?
4648         {\bbbl@ifunset{\bbbl@##1dflt@}% 2=F - (3) from generic?
4649           {}}% 123=F - nothing!
4650         {\bbbl@exp{% 3=T - from generic
4651           \global\let\<\bbbl@##1dflt@\language name>%
4652           \<\bbbl@##1dflt@>}}}%
4653         {\bbbl@exp{% 2=T - from script
4654           \global\let\<\bbbl@##1dflt@\language name>%
4655           \<\bbbl@##1dflt@*\bbbl@tempa>}}}%
4656       {}}% 1=T - language, already defined
4657 \def\bbbl@tempa{\bbbl@nostdfont{}}% TODO. Don't use \bbbl@tempa

```

```

4658 \bbl@foreach\bbl@font@fams{%      don't gather with prev for
4659   \bbl@ifunset{\bbl@##ldflt@\language\name}%
4660   {\bbl@cs{famrst@##1}%
4661    \global\bbl@csarg\let{famrst@##1}\relax}%
4662   {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4663    \\bbl@add\\originalTeX{%
4664     \\bbl@font@rst{\bbl@cl{##ldflt}}%
4665     \<##ldfault>\<##lfamily>{##1}}%
4666     \\bbl@font@set{\bbl@##ldflt@\language\name}% the main part!
4667     \<##ldfault>\<##lfamily>}}}%
4668 \bbl@ifrestoring{ }\bbl@tempa}%

```

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

```

4669 \ifx\fbfamily\undefined\else      % if latex
4670 \ifcase\bbl@engine                  % if pdftex
4671   \let\bbl@ckeckstdfonts\relax
4672 \else
4673   \def\bbl@ckeckstdfonts{%
4674     \begingroup
4675     \global\let\bbl@ckeckstdfonts\relax
4676     \let\bbl@tempa\empty
4677     \bbl@foreach\bbl@font@fams{%
4678       \bbl@ifunset{\bbl@##ldflt@}%
4679       {\@nameuse{##lfamily}%
4680        \bbl@csarg\gdef{WFF@\fbfamily}}}% Flag
4681       \bbl@exp{\\bbl@add\\bbl@tempa{* \<##lfamily>= \fbfamily\\}%
4682        \space\space\fontname\font\\}%
4683       \bbl@csarg\xdef{##ldflt@}{\fbfamily}%
4684       \expandafter\xdef\csname ##ldfault\endcsname{\fbfamily}}}%
4685       {}}%
4686   \ifx\bbl@tempa\empty\else
4687     \bbl@infowarn{The following font families will use the default\\%
4688       settings for all or some languages:\\%
4689       \bbl@tempa
4690       There is nothing intrinsically wrong with it, but\\%
4691       'babel' will no set Script and Language, which could\\%
4692       be relevant in some languages. If your document uses\\%
4693       these families, consider redefining them with \string\babelfont.\\%
4694       Reported}%
4695   \fi
4696 \endgroup}
4697 \fi
4698 \fi

```

Now the macros defining the font with `fontspec`.

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

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

```

4699 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4700   \bbl@xin@{<>}{#1}%
4701   \ifin@
4702     \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4703   \fi
4704   \bbl@exp{%
4705     \def\\#2{#1}%          eg, \rmdefault{\bbl@rmdflt@lang}
4706     \\bbl@ifsamestring{#2}{\fbfamily}%
4707     {\\#3%

```

```

4708     \\\bbl@ifsamestring{\f@series}{\bfdefault}{\bfseries}}}%
4709     \let\\bbl@tempa\relax}%
4710     {}}}}
4711 %     TODO - next should be global?, but even local does its job. I'm
4712 %     still not sure -- must investigate:
4713 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4714     \let\bbl@tempe\bbl@mapselect
4715     \edef\bbl@tempb{\bbl@stripslash#4/}% Catcodes hack (better pass it).
4716     \bbl@exp{\bbl@replace\\bbl@tempb{\bbl@stripslash\family/}}}%
4717     \let\bbl@mapselect\relax
4718     \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4719     \let#4\empty % Make sure \renewfontfamily is valid
4720     \bbl@exp{%
4721         \let\\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
4722         <\keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4723         {\bbl@newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4724         <\keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4725         {\bbl@newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4726         \let\\bbl@tempfs@nx<__fontspec_warning:nx>%
4727         \let<__fontspec_warning:nx>\\bbl@fs@warn@nx
4728         \let\\bbl@tempfs@nxx<__fontspec_warning:nxx>%
4729         \let<__fontspec_warning:nxx>\\bbl@fs@warn@nxx
4730         \\renewfontfamily\\#4%
4731         [\bbl@cl{lsys},%
4732         \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
4733         #2}}{#3}% ie \bbl@exp{..}{#3}
4734     \bbl@exp{%
4735         \let<__fontspec_warning:nx>\\bbl@tempfs@nx
4736         \let<__fontspec_warning:nxx>\\bbl@tempfs@nxx}%
4737     \begingroup
4738     #4%
4739     \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4740     \endgroup % TODO. Find better tests:
4741     \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4742     {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
4743     \ifin@
4744     \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
4745     \fi
4746     \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4747     {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
4748     \ifin@
4749     \global\bbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
4750     \fi
4751     \let#4\bbl@temp@fam
4752     \bbl@exp{\let<\bbl@stripslash#4\space>}\bbl@temp@pfam
4753     \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.

```

4754 \def\bbl@font@rst#1#2#3#4{%
4755     \bbl@csarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}

```

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

```

4756 \def\bbl@font@fams{rm,sf,tt}
4757 <</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.

```

4758 <<{*Footnote changes}>> ≡
4759 \bbl@trace{Bidi footnotes}
4760 \ifnum\bbl@bidimode>\z@ % Any bidi=
4761 \def\bbl@footnote#1#2#3{%
4762   \@ifnextchar[%
4763     {\bbl@footnote@o{#1}{#2}{#3}}%
4764     {\bbl@footnote@x{#1}{#2}{#3}}}
4765 \long\def\bbl@footnote@x#1#2#3#4{%
4766   \bgroup
4767   \select@language@x{\bbl@main@language}%
4768   \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4769   \egroup}
4770 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4771   \bgroup
4772   \select@language@x{\bbl@main@language}%
4773   \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4774   \egroup}
4775 \def\bbl@footnotetext#1#2#3{%
4776   \@ifnextchar[%
4777     {\bbl@footnotetext@o{#1}{#2}{#3}}%
4778     {\bbl@footnotetext@x{#1}{#2}{#3}}}
4779 \long\def\bbl@footnotetext@x#1#2#3#4{%
4780   \bgroup
4781   \select@language@x{\bbl@main@language}%
4782   \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4783   \egroup}
4784 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4785   \bgroup
4786   \select@language@x{\bbl@main@language}%
4787   \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4788   \egroup}
4789 \def\BabelFootnote#1#2#3#4{%
4790   \ifx\bbl@fn@footnote\@undefined
4791     \let\bbl@fn@footnote\footnote
4792   \fi
4793   \ifx\bbl@fn@footnotetext\@undefined
4794     \let\bbl@fn@footnotetext\footnotetext
4795   \fi
4796   \bbl@ifblank{#2}%
4797     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4798     \@namedef{\bbl@stripslash#1text}%
4799     {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4800     {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4801     \@namedef{\bbl@stripslash#1text}%
4802     {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4803 \fi
4804 <</Footnote changes>>

```

Now, the code.

```

4805 <{*xetex}>
4806 \def\BabelStringsDefault{unicode}
4807 \let\xebbl@stop\relax
4808 \AddBabelHook{xetex}{encodedcommands}{%
4809   \def\bbl@tempa{#1}%
4810   \ifx\bbl@tempa\@empty
4811     \XeTeXinputencoding"bytes"%
4812   \else
4813     \XeTeXinputencoding"#1"%
4814   \fi
4815   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4816 \AddBabelHook{xetex}{stopcommands}{%
4817   \xebbl@stop
4818   \let\xebbl@stop\relax}

```

```

4819 \def\bb@intraspace#1 #2 #3\@@{%
4820   \bb@csarg\gdef{xeisp@\language\name}%
4821   {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4822 \def\bb@intrapenalty#1\@@{%
4823   \bb@csarg\gdef{xeipn@\language\name}%
4824   {\XeTeXlinebreakpenalty #1\relax}}
4825 \def\bb@provide@intraspace{%
4826   \bb@xin@{/s}{/\bb@cl{\lnbrk}}}%
4827   \ifin@ \else \bb@xin@{/c}{/\bb@cl{\lnbrk}} \fi
4828   \ifin@
4829     \bb@ifunset{\bb@intsp@\language\name}{\}%
4830     {\expandafter\ifx\csname \bb@intsp@\language\name\endcsname\@empty\else
4831       \ifx\bb@KVP@intraspace\@nnil
4832         \bb@exp{%
4833           \bb@intraspace\bb@cl{\intsp}\@@}%
4834         \fi
4835         \ifx\bb@KVP@intrapenalty\@nnil
4836           \bb@intrapenalty0\@@
4837         \fi
4838       \fi
4839       \ifx\bb@KVP@intraspace\@nnil\else % We may override the ini
4840         \expandafter\bb@intraspace\bb@KVP@intraspace\@@
4841       \fi
4842       \ifx\bb@KVP@intrapenalty\@nnil\else
4843         \expandafter\bb@intrapenalty\bb@KVP@intrapenalty\@@
4844       \fi
4845       \bb@exp{%
4846         % TODO. Execute only once (but redundant):
4847         \bb@add\<extras\language\name>{%
4848           \XeTeXlinebreaklocale "\bb@cl{\tbcpr}"%
4849           \<\bb@xeisp@\language\name>%
4850           \<\bb@xeipn@\language\name>}%
4851         \bb@toglobal\<extras\language\name>%
4852         \bb@add\<noextras\language\name>{%
4853           \XeTeXlinebreaklocale ""}%
4854         \bb@toglobal\<noextras\language\name>%
4855       \ifx\bb@ispace\@undefined
4856         \gdef\bb@ispace{\bb@cl{\xeisp}}%
4857       \ifx\AtBeginDocument\@notprerr
4858         \expandafter\@secondoftwo % to execute right now
4859       \fi
4860       \AtBeginDocument{\bb@patchfont{\bb@ispace}}%
4861     \fi}%
4862 \fi}
4863 \ifx\DisableBabelHook\@undefined\endinput\fi
4864 \AddBabelHook{babel-fontspec}{afterextras}{\bb@switchfont}
4865 \AddBabelHook{babel-fontspec}{beforestart}{\bb@ccheckstdfonts}
4866 \DisableBabelHook{babel-fontspec}
4867 \langle Font selection \rangle
4868 \def\bb@provide@extra#1{}

```

10 Support for interchar

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

```

4869 \ifnum\Xe@alloc@intercharclass<\thr@@
4870   \Xe@alloc@intercharclass\thr@@
4871 \fi
4872 \chardef\bb@xeclass@default=\z@
4873 \chardef\bb@xeclass@cjkideogram=\@ne
4874 \chardef\bb@xeclass@cjkleftpunctuation=\tw@
4875 \chardef\bb@xeclass@cjkrightpunctuation=\thr@@

```



```

4876 \chardef\bbl@xeclass@boundary@=4095
4877 \chardef\bbl@xeclass@ignore@=4096

```

The machinery is activated with a hook (enabled only if actually used). Here `\bbl@tempc` is pre-set with `\bbl@usingxeclass`, defined below. The standard mechanism based on `\originalTeX` to save, set and restore values is used. `\count@` stores the previous char to be set, except at the beginning (0) and after `\bbl@upto`, which is the previous char negated, as a flag to mark a range.

```

4878 \AddBabelHook{babel-interchar}{beforeextras}{%
4879   \@nameuse{bbl@xechars@\languagename}}
4880 \DisableBabelHook{babel-interchar}
4881 \protected\def\bbl@charclass#1{%
4882   \ifnum\count@<\z@
4883     \count@-\count@
4884     \loop
4885       \bbl@exp{%
4886         \\babel@savevariable{\XeTeXcharclass`\Uchar\count@}}%
4887         \XeTeXcharclass\count@ \bbl@tempc
4888         \ifnum\count@<`#1\relax
4889           \advance\count@\@ne
4890         \repeat
4891   \else
4892     \babel@savevariable{\XeTeXcharclass`#1}%
4893     \XeTeXcharclass`#1 \bbl@tempc
4894   \fi
4895   \count@`#1\relax}

```

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

```

4896 \newcommand\IfBabelIntercharT[1]{%
4897   \let\bbl@tempa@gobble % Assume to ignore
4898   \edef\bbl@tempb{\zap@space#1 \@empty}%
4899   \ifx\bbl@KVP@interchar\@nnil\else
4900     \bbl@replace\bbl@KVP@interchar{ }{,}%
4901     \bbl@foreach\bbl@tempb{%
4902       \bbl@xin@{,##1,}{, \bbl@KVP@interchar,%}
4903       \ifin@
4904         \let\bbl@tempa\@firstofone
4905       \fi}%
4906   \fi
4907   \bbl@tempa}
4908 \newcommand\babelcharclass[3]{%
4909   \EnableBabelHook{babel-interchar}%
4910   \bbl@csarg\newXeTeXintercharclass{xeclass@#2@#1}%
4911   \def\bbl@tempb##1{%
4912     \ifx##1\@empty\else
4913       \ifx##1-
4914         \bbl@upto
4915       \else
4916         \bbl@charclass{%
4917           \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
4918       \fi
4919       \expandafter\bbl@tempb
4920     \fi}%
4921   \bbl@ifunset{bbl@xechars@#1}%
4922   {\toks@{%
4923     \babel@savevariable\XeTeXinterchartokenstate
4924     \XeTeXinterchartokenstate\@ne
4925   }}%
4926   {\toks@\expandafter\expandafter\expandafter{%
4927     \csname bbl@xechars@#1\endcsname}}}%

```

```

4928 \bbl@csarg\edef{xechars@#1}{%
4929   \the\toks@
4930   \bbl@usingxeclass\csname bbl@xeclass@#2@#1\endcsname
4931   \bbl@tempb#3\@empty}}
4932 \protected\def\bbl@usingxeclass#1{\count@ \z@ \let\bbl@tempc#1}
4933 \protected\def\bbl@upto{%
4934   \ifnum\count@>\z@
4935     \advance\count@\@ne
4936     \count@-\count@
4937   \else\ifnum\count@=\z@
4938     \bbl@charclass{-}%
4939   \else
4940     \bbl@error{Double hyphens aren't allowed in \string\babelcharclass\\%
4941               because it's potentially ambiguous}%
4942     {See the manual for further info}%
4943   \fi\fi}

```

And finally, the command with the code to be inserted. If the language doesn't define a class, then use the global one, as defined above. For the definition there is a intermediate macro, which can be 'disabled' with `\bbl@ic@<label><lang>`.

```

4944 \newcommand\babelinterchar[5][{}]{%
4945   \let\bbl@kv@label\@empty
4946   \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}%
4947   \@namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
4948   {\ifnum\language=\l@nohyphenation
4949     \expandafter\@gobble
4950   \else
4951     \expandafter\@firstofone
4952   \fi
4953   {#5}}%
4954   \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone
4955   \bbl@exp{\bbl@for\bbl@tempa{\zap@space#3 \@empty}}%
4956   \bbl@exp{\bbl@for\bbl@tempb{\zap@space#4 \@empty}}%
4957   \XeTeXinterchartoks
4958     \@nameuse{bbl@xeclass@\bbl@tempa @}%
4959     \bbl@ifunset{bbl@xeclass@\bbl@tempa @#2}{#2} %
4960     \@nameuse{bbl@xeclass@\bbl@tempb @}%
4961     \bbl@ifunset{bbl@xeclass@\bbl@tempb @#2}{#2} %
4962     = \expandafter{%
4963       \csname bbl@ic@\bbl@kv@label @#2\expandafter\endcsname
4964       \csname\zap@space bbl@xeinter@\bbl@kv@label
4965         @#3@#4@#2 \@empty\endcsname}}}}
4966 \DeclareRobustCommand\enablelocaleinterchar[1]{%
4967   \bbl@ifunset{bbl@ic@#1@\language}%
4968   {\bbl@error
4969     {'#1' for '\language' cannot be enabled.\\%
4970     Maybe there is a typo.}%
4971     {See the manual for further details.}}%
4972   {\bbl@csarg\let{ic@#1@\language}\@firstofone}}
4973 \DeclareRobustCommand\disablelocaleinterchar[1]{%
4974   \bbl@ifunset{bbl@ic@#1@\language}%
4975   {\bbl@error
4976     {'#1' for '\language' cannot be disabled.\\%
4977     Maybe there is a typo.}%
4978     {See the manual for further details.}}%
4979   {\bbl@csarg\let{ic@#1@\language}\@gobble}}
4980 </xetex>

```

10.1 Layout

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

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

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

```

4981 (*xetex | texxet)
4982 \providecommand\bbbl@provide@intraspace{}
4983 \bbbl@trace{Redefinitions for bidi layout}
4984 \def\bbbl@sspre@caption{%
4985   \bbbl@exp{\everyhbox{\bbbl@textdir\bbbl@cs{wdir@\bbbl@main@language}}}}
4986 \ifx\bbbl@opt@layout\@nnil\else % if layout=..
4987 \def\bbbl@startskip{\ifcase\bbbl@thepardir\leftskip\else\rightskip\fi}
4988 \def\bbbl@endskip{\ifcase\bbbl@thepardir\rightskip\else\leftskip\fi}
4989 \ifx\bbbl@beforeforeign\leavevmode % A poor test for bidi=
4990 \def\@hangfrom#1{%
4991   \setbox\@tempboxa\hbox{#1}%
4992   \hangindent\ifcase\bbbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4993   \noindent\box\@tempboxa}
4994 \def\raggedright{%
4995   \let\@centercr
4996   \bbbl@startskip\z@skip
4997   \@rightskip\@flushglue
4998   \bbbl@endskip\@rightskip
4999   \parindent\z@
5000   \parfillskip\bbbl@startskip}
5001 \def\raggedleft{%
5002   \let\@centercr
5003   \bbbl@startskip\@flushglue
5004   \bbbl@endskip\z@skip
5005   \parindent\z@
5006   \parfillskip\bbbl@endskip}
5007 \fi
5008 \IfBabelLayout{lists}
5009 {\bbbl@sreplace\list
5010   {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbbl@listleftmargin}%
5011   \def\bbbl@listleftmargin{%
5012     \ifcase\bbbl@thepardir\leftmargin\else\rightmargin\fi}%
5013   \ifcase\bbbl@engine
5014     \def\labelenumii{}\theenumii}% pdfTeX doesn't reverse ()
5015     \def\p@enumiii{\p@enumii}\theenumii}%
5016   \fi
5017   \bbbl@sreplace\@verbatim
5018     {\leftskip\@totalleftmargin}%
5019     {\bbbl@startskip\textwidth
5020       \advance\bbbl@startskip-\linewidth}%
5021   \bbbl@sreplace\@verbatim
5022     {\rightskip\z@skip}%
5023     {\bbbl@endskip\z@skip}}%
5024 {}
5025 \IfBabelLayout{contents}
5026 {\bbbl@sreplace\@dottedtocline{\leftskip}{\bbbl@startskip}%
5027   \bbbl@sreplace\@dottedtocline{\rightskip}{\bbbl@endskip}}
5028 {}
5029 \IfBabelLayout{columns}
5030 {\bbbl@sreplace\@outputdblcol{\hbext@\textwidth}{\bbbl@outputbox}%
5031   \def\bbbl@outputbox#1{%
5032     \hbext@\textwidth%
5033     \hskip\columnwidth
5034     \hfil
5035     {\normalcolor\vrule \@width\columnseprule}%
5036     \hfil
5037     \hbext@\columnwidth{\box\@leftcolumn \hss}%
5038     \hskip-\textwidth
5039     \hbext@\columnwidth{\box\@outputbox \hss}%
5040     \hskip\columnsep

```

```

5041 \hskip\columnwidth}}}%
5042 {}
5043 <<Footnote changes>>
5044 \IfBabelLayout{footnotes}%
5045 {\BabelFootnote\footnote\languagename{}}}%
5046 \BabelFootnote\localfootnote\languagename{}}}%
5047 \BabelFootnote\mainfootnote{}}{}%
5048 {}

```

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.

```

5049 \IfBabelLayout{counters*}%
5050 {\bbl@add\bbl@opt@layout{.counters.}%
5051 \AddToHook{shipout/before}{%
5052 \let\bbl@tempa\babelsublr
5053 \let\babelsublr\@firstofone
5054 \let\bbl@save@thepage\thepage
5055 \protected@edef\thepage{\thepage}%
5056 \let\babelsublr\bbl@tempa}%
5057 \AddToHook{shipout/after}{%
5058 \let\thepage\bbl@save@thepage}}{}%
5059 \IfBabelLayout{counters}%
5060 {\let\bbl@latinarabic=\@arabic
5061 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
5062 \let\bbl@asciroman=\@roman
5063 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5064 \let\bbl@asciiRoman=\@Roman
5065 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}%
5066 \fi % end if layout
5067 </xetex | texpet>

```

10.2 8-bit TeX

Which start just above, because some code is shared with xetex. Now, 8-bit specific stuff. If just one encoding has been declared, then assume no switching is necessary (1).

```

5068 <*texpet>
5069 \def\bbl@provide@extra#1{%
5070 % == auto-select encoding ==
5071 \ifx\bbl@encoding@select@off\@empty\else
5072 \bbl@ifunset{\bbl@encoding@#1}%
5073 {\def\@elt##1{,##1,}%
5074 \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5075 \count@\z@
5076 \bbl@foreach\bbl@tempe{%
5077 \def\bbl@tempd{##1}% Save last declared
5078 \advance\count@\@ne}%
5079 \ifnum\count@>\@ne % (1)
5080 \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5081 \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
5082 \bbl@replace\bbl@tempa{ },}%
5083 \global\bbl@csarg\let{encoding@#1}\@empty
5084 \bbl@xin@{\bbl@tempd,}{,\bbl@tempa,}%
5085 \ifin\@else % if main encoding included in ini, do nothing
5086 \let\bbl@tempb\relax
5087 \bbl@foreach\bbl@tempa{%
5088 \ifx\bbl@tempb\relax
5089 \bbl@xin@{,##1,}{,\bbl@tempe,}%
5090 \ifin\@def\bbl@tempb{##1}\fi
5091 \fi}%
5092 \ifx\bbl@tempb\relax\else
5093 \bbl@exp{%
5094 \global\<\bbl@add>\<\bbl@preextras@#1>\<\bbl@encoding@#1>%
5095 \gdef\<\bbl@encoding@#1>{%

```

```

5096          \\babel@save\\f@encoding
5097          \\bbl@add\\originalTeX{\\selectfont}%
5098          \\fontencoding{\\bbl@tempb}%
5099          \\selectfont}}}%
5100      \fi
5101      \fi
5102      \fi}%
5103  {}%
5104  \fi}
5105 </texxet>

```

10.3 LuaTeX

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

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

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

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

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

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

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

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

```

5106 <{*luatex}
5107 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
5108 \bbl@trace{Read language.dat}
5109 \ifx\bbl@readstream\undefined
5110   \csname newread\endcsname\bbl@readstream
5111 \fi
5112 \begingroup
5113   \toks@{}
5114   \count@z@ % 0=start, 1=0th, 2=normal
5115   \def\bbl@process@line#1#2 #3 #4 {%
5116     \ifx=#1%
5117       \bbl@process@synonym{#2}%
5118     \else
5119       \bbl@process@language{#1#2}{#3}{#4}%
5120     \fi
5121     \ignorespaces}
5122   \def\bbl@manylang{%
5123     \ifnum\bbl@last>\@ne

```

```

5124 \bbl@info{Non-standard hyphenation setup}%
5125 \fi
5126 \let\bbl@manylang\relax
5127 \def\bbl@process@language#1#2#3{%
5128 \ifcase\count@
5129 \ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
5130 \or
5131 \count@\tw@
5132 \fi
5133 \ifnum\count@=\tw@
5134 \expandafter\addlanguage\csname l@#1\endcsname
5135 \language\allocationnumber
5136 \chardef\bbl@last\allocationnumber
5137 \bbl@manylang
5138 \let\bbl@elt\relax
5139 \xdef\bbl@languages{%
5140 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
5141 \fi
5142 \the\toks@
5143 \toks@{}}
5144 \def\bbl@process@synonym@aux#1#2{%
5145 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
5146 \let\bbl@elt\relax
5147 \xdef\bbl@languages{%
5148 \bbl@languages\bbl@elt{#1}{#2}{}}}%
5149 \def\bbl@process@synonym#1{%
5150 \ifcase\count@
5151 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
5152 \or
5153 \ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}}%
5154 \else
5155 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
5156 \fi}
5157 \ifx\bbl@languages\undefined % Just a (sensible?) guess
5158 \chardef\l@english\z@
5159 \chardef\l@USenglish\z@
5160 \chardef\bbl@last\z@
5161 \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
5162 \gdef\bbl@languages{%
5163 \bbl@elt{english}{0}{hyphen.tex}}%
5164 \bbl@elt{USenglish}{0}{}}
5165 \else
5166 \global\let\bbl@languages@format\bbl@languages
5167 \def\bbl@elt#1#2#3#4{% Remove all except language 0
5168 \ifnum#2>\z@ \else
5169 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
5170 \fi}%
5171 \xdef\bbl@languages{\bbl@languages}%
5172 \fi
5173 \def\bbl@elt#1#2#3#4{\@namedef{zth#1}} % Define flags
5174 \bbl@languages
5175 \openin\bbl@readstream=language.dat
5176 \ifEOF\bbl@readstream
5177 \bbl@warning{I couldn't find language.dat. No additional\\%
5178 patterns loaded. Reported}%
5179 \else
5180 \loop
5181 \endlinechar\m@ne
5182 \read\bbl@readstream to \bbl@line
5183 \endlinechar\^M
5184 \if T\ifEOF\bbl@readstream F\fi T\relax
5185 \ifx\bbl@line\empty\else
5186 \edef\bbl@line{\bbl@line\space\space\space}%

```

```

5187         \expandafter\bb@l@process@line\bb@l@line\relax
5188     \fi
5189 \repeat
5190 \fi
5191 \closein\bb@l@readstream
5192 \endgroup
5193 \bb@l@trace{Macros for reading patterns files}
5194 \def\bb@l@get@enc#1:#2:#3\@@@{\def\bb@l@hyph@enc{#2}}
5195 \ifx\babelcatcodetablenum\undefined
5196     \ifx\newcatcodetable\undefined
5197         \def\babelcatcodetablenum{5211}
5198         \def\bb@l@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5199     \else
5200         \newcatcodetable\babelcatcodetablenum
5201         \newcatcodetable\bb@l@pattcodes
5202     \fi
5203 \else
5204     \def\bb@l@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5205 \fi
5206 \def\bb@l@luapatterns#1#2{%
5207     \bb@l@get@enc#1::\@@@
5208     \setbox\z@\hbox\bgroup
5209         \begingroup
5210             \savecatcodetable\babelcatcodetablenum\relax
5211             \initcatcodetable\bb@l@pattcodes\relax
5212             \catcodetable\bb@l@pattcodes\relax
5213             \catcode\#=6 \catcode\$=3 \catcode\&=4 \catcode\^=7
5214             \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\~=13
5215             \catcode\@=11 \catcode\^^I=10 \catcode\^^J=12
5216             \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
5217             \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
5218             \catcode\`=12 \catcode\'=12 \catcode\"=12
5219             \input #1\relax
5220             \catcodetable\babelcatcodetablenum\relax
5221         \endgroup
5222     \def\bb@l@tempa{#2}%
5223     \ifx\bb@l@tempa\@empty\else
5224         \input #2\relax
5225     \fi
5226 \egroup}%
5227 \def\bb@l@patterns@lua#1{%
5228     \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5229         \csname l@#1\endcsname
5230         \edef\bb@l@tempa{#1}%
5231     \else
5232         \csname l@#1:\f@encoding\endcsname
5233         \edef\bb@l@tempa{#1:\f@encoding}%
5234     \fi\relax
5235     \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5236     \@ifundefined{bb@l@hyphendata@the\language}%
5237     {\def\bb@l@elt##1##2##3##4{%
5238         \ifnum##2=\csname l@bb@l@tempa\endcsname % #2=spanish, dutch:OT1...
5239         \def\bb@l@tempb{##3}%
5240         \ifx\bb@l@tempb\@empty\else % if not a synonymous
5241             \def\bb@l@tempc{{##3}{##4}}%
5242         \fi
5243         \bb@l@csarg\xdef{hyphendata@##2}{\bb@l@tempc}%
5244         \fi}%
5245     \bb@l@languages
5246     \@ifundefined{bb@l@hyphendata@the\language}%
5247     {\bb@l@info{No hyphenation patterns were set for\\%
5248         language '\bb@l@tempa'. Reported}}%
5249     {\expandafter\expandafter\expandafter\bb@l@luapatterns

```

```

5250         \csname bbl@hyphendata@the\language\endcsname}}{}}
5251 \endinput\fi
5252 % Here ends \ifx\AddBabelHook\@undefined
5253 % A few lines are only read by hyphen.cfg
5254 \ifx\DisableBabelHook\@undefined
5255   \AddBabelHook{luatex}{everylanguage}{%
5256     \def\process@language##1##2##3{%
5257       \def\process@line####1####2 ####3 ####4 {}}}
5258   \AddBabelHook{luatex}{loadpatterns}{%
5259     \input #1\relax
5260     \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5261       {#{1}}{}}}
5262   \AddBabelHook{luatex}{loadexceptions}{%
5263     \input #1\relax
5264     \def\bbl@tempb##1##2{{#{1}}{#{1}}}%
5265     \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5266       {\expandafter\expandafter\expandafter\bbl@tempb
5267         \csname bbl@hyphendata@the\language\endcsname}}
5268 \endinput\fi
5269 % Here stops reading code for hyphen.cfg
5270 % The following is read the 2nd time it's loaded
5271 \begingroup % TODO - to a lua file
5272 \catcode`\%=12
5273 \catcode`\'=12
5274 \catcode`\=12
5275 \catcode`\:=12
5276 \directlua{
5277   Babel = Babel or {}
5278   function Babel.bytes(line)
5279     return line:gsub(".",
5280       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5281   end
5282   function Babel.begin_process_input()
5283     if luatexbase and luatexbase.add_to_callback then
5284       luatexbase.add_to_callback('process_input_buffer',
5285         Babel.bytes, 'Babel.bytes')
5286     else
5287       Babel.callback = callback.find('process_input_buffer')
5288       callback.register('process_input_buffer', Babel.bytes)
5289     end
5290   end
5291   function Babel.end_process_input ()
5292     if luatexbase and luatexbase.remove_from_callback then
5293       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5294     else
5295       callback.register('process_input_buffer', Babel.callback)
5296     end
5297   end
5298   function Babel.addpatterns(pp, lg)
5299     local lg = lang.new(lg)
5300     local pats = lang.patterns(lg) or ''
5301     lang.clear_patterns(lg)
5302     for p in pp:gmatch('[^%s]+') do
5303       ss = ''
5304       for i in string.utfcharacters(p:gsub('%d', '')) do
5305         ss = ss .. '%d?' .. i
5306       end
5307       ss = ss:gsub('^%d%?%', '%%.') .. '%d?'
5308       ss = ss:gsub('%.%d%?$', '%%.')
5309       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5310       if n == 0 then
5311         tex.sprint(
5312           [[\string\csname\space bbl@info\endcsname{New pattern: }]]

```



```

5313         .. p .. [[]])
5314         pats = pats .. ' ' .. p
5315     else
5316         tex.sprint(
5317             [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5318             .. p .. [[]])
5319     end
5320 end
5321 lang.patterns(lg, pats)
5322 end
5323 Babel.characters = Babel.characters or {}
5324 Babel.ranges = Babel.ranges or {}
5325 function Babel.hlist_has_bidi(head)
5326     local has_bidi = false
5327     local ranges = Babel.ranges
5328     for item in node.traverse(head) do
5329         if item.id == node.id'glyph' then
5330             local itemchar = item.char
5331             local chardata = Babel.characters[itemchar]
5332             local dir = chardata and chardata.d or nil
5333             if not dir then
5334                 for nn, et in ipairs(ranges) do
5335                     if itemchar < et[1] then
5336                         break
5337                     elseif itemchar <= et[2] then
5338                         dir = et[3]
5339                         break
5340                     end
5341                 end
5342             end
5343             if dir and (dir == 'al' or dir == 'r') then
5344                 has_bidi = true
5345             end
5346         end
5347     end
5348     return has_bidi
5349 end
5350 function Babel.set_chranges_b (script, chrng)
5351     if chrng == '' then return end
5352     texio.write('Replacing ' .. script .. ' script ranges')
5353     Babel.script_blocks[script] = {}
5354     for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5355         table.insert(
5356             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5357     end
5358 end
5359 function Babel.discard_sublrl(str)
5360     if str:find( [[\string\indexentry]] ) and
5361        str:find( [[\string\babelsublrl]] ) then
5362         str = str:gsub( [[\string\babelsublrls*(%b{})]],
5363             function(m) return m:sub(2,-2) end )
5364     end
5365     return str
5366 end
5367 }
5368 \endgroup
5369 \ifx\newattribute\@undefined\else % Test for plain
5370     \newattribute\bbl@attr@locale
5371     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5372     \AddBabelHook{luatex}{beforeextras}{%
5373         \setattribute\bbl@attr@locale\localeid}
5374 \fi
5375 \def\BabelStringsDefault{unicode}

```

```

5376 \let\luabbl@stop\relax
5377 \AddBabelHook{luatex}{encodedcommands}{%
5378   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5379   \ifx\bbl@tempa\bbl@tempb\else
5380     \directlua{Babel.begin_process_input()}%
5381     \def\luabbl@stop{%
5382       \directlua{Babel.end_process_input()}}%
5383   \fi}%
5384 \AddBabelHook{luatex}{stopcommands}{%
5385   \luabbl@stop
5386   \let\luabbl@stop\relax}
5387 \AddBabelHook{luatex}{patterns}{%
5388   \@ifundefined{bbl@hyphendata@the\language}%
5389     {\def\bbl@elt##1##2##3##4{%
5390       \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5391       \def\bbl@tempb{##3}%
5392       \ifx\bbl@tempb\@empty\else % if not a synonymous
5393         \def\bbl@tempc{{##3}{##4}}%
5394       \fi
5395       \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5396     \fi}%
5397   \bbl@languages
5398   \@ifundefined{bbl@hyphendata@the\language}%
5399     {\bbl@info{No hyphenation patterns were set for\\%
5400       language '#2'. Reported}}%
5401     {\expandafter\expandafter\expandafter\bbl@luapatterns
5402       \csname bbl@hyphendata@the\language\endcsname}}}%
5403   \@ifundefined{bbl@patterns@}{}%
5404   \begingroup
5405     \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5406     \ifin@else
5407       \ifx\bbl@patterns@\@empty\else
5408         \directlua{ Babel.addpatterns(
5409           [[\bbl@patterns@]], \number\language) }%
5410       \fi
5411       \@ifundefined{bbl@patterns@#1}%
5412       \@empty
5413       {\directlua{ Babel.addpatterns(
5414         [[\space\csname bbl@patterns@#1\endcsname]],
5415         \number\language) }}%
5416       \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5417     \fi
5418   \endgroup}%
5419   \bbl@exp{%
5420     \bbl@ifunset{bbl@prehc@languagename}{}%
5421     {\bbl@ifblank{\bbl@cs{prehc@languagename}}}%
5422     {\prehyphenchar=\bbl@cl{prehc}\relax}}}

```

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

```

5423 \@onlypreamble\babelpatterns
5424 \AtEndOfPackage{%
5425   \newcommand\babelpatterns[2][\@empty]{%
5426     \ifx\bbl@patterns@\relax
5427       \let\bbl@patterns@\@empty
5428     \fi
5429     \ifx\bbl@pttnlist@\@empty\else
5430       \bbl@warning{%
5431         You must not intermingle \string\selectlanguage\space and\\%
5432         \string\babelpatterns\space or some patterns will not\\%
5433         be taken into account. Reported}%
5434       \fi

```

```

5435 \ifx\@empty#1%
5436 \protected@edef\bbl@patterns@\bbl@patterns@space#2}%
5437 \else
5438 \edef\bbl@tempb{\zap@space#1 \@empty}%
5439 \bbl@for\bbl@tempa\bbl@tempb{%
5440 \bbl@fixname\bbl@tempa
5441 \bbl@iflanguage\bbl@tempa{%
5442 \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5443 \ifundefined\bbl@patterns@\bbl@tempa}%
5444 \@empty
5445 {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5446 #2}}}%
5447 \fi}}

```

10.4 Southeast Asian scripts

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

```

5448% TODO - to a lua file
5449 \directlua{
5450   Babel = Babel or {}
5451   Babel.linebreaking = Babel.linebreaking or {}
5452   Babel.linebreaking.before = {}
5453   Babel.linebreaking.after = {}
5454   Babel.locale = {} % Free to use, indexed by \localeid
5455   function Babel.linebreaking.add_before(func, pos)
5456     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5457     if pos == nil then
5458       table.insert(Babel.linebreaking.before, func)
5459     else
5460       table.insert(Babel.linebreaking.before, pos, func)
5461     end
5462   end
5463   function Babel.linebreaking.add_after(func)
5464     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5465     table.insert(Babel.linebreaking.after, func)
5466   end
5467 }
5468 \def\bbl@intraspace#1 #2 #3\@@{%
5469 \directlua{
5470   Babel = Babel or {}
5471   Babel.intraspaces = Babel.intraspaces or {}
5472   Babel.intraspaces['\csname bbl@sbcpr@language\endcsname'] = %
5473     {b = #1, p = #2, m = #3}
5474   Babel.locale_props[\the\localeid].intraspace = %
5475     {b = #1, p = #2, m = #3}
5476 }}
5477 \def\bbl@intrapenalty#1\@@{%
5478 \directlua{
5479   Babel = Babel or {}
5480   Babel.intrapenalties = Babel.intrapenalties or {}
5481   Babel.intrapenalties['\csname bbl@sbcpr@language\endcsname'] = #1
5482   Babel.locale_props[\the\localeid].intrapenalty = #1
5483 }}
5484 \begingroup
5485 \catcode`\%=12
5486 \catcode`\^=14
5487 \catcode`\'=12
5488 \catcode`\~=12
5489 \gdef\bbl@seaintraspace{^
5490 \let\bbl@seaintraspace\relax

```

```

5491 \directlua{
5492   Babel = Babel or {}
5493   Babel.sea_enabled = true
5494   Babel.sea_ranges = Babel.sea_ranges or {}
5495   function Babel.set_chrngs (script, chrng)
5496     local c = 0
5497     for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5498       Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5499       c = c + 1
5500     end
5501   end
5502   function Babel.sea_disc_to_space (head)
5503     local sea_ranges = Babel.sea_ranges
5504     local last_char = nil
5505     local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5506     for item in node.traverse(head) do
5507       local i = item.id
5508       if i == node.id'glyph' then
5509         last_char = item
5510       elseif i == 7 and item.subtype == 3 and last_char
5511         and last_char.char > 0x0C99 then
5512         quad = font.getfont(last_char.font).size
5513         for lg, rg in pairs(sea_ranges) do
5514           if last_char.char > rg[1] and last_char.char < rg[2] then
5515             lg = lg:sub(1, 4) ^% Remove trailing number of, eg, Cyril1
5516             local intraspace = Babel.intraspaces[lg]
5517             local intrapenalty = Babel.intrapenalties[lg]
5518             local n
5519             if intrapenalty ~= 0 then
5520               n = node.new(14, 0)      ^% penalty
5521               n.penalty = intrapenalty
5522               node.insert_before(head, item, n)
5523             end
5524             n = node.new(12, 13)      ^% (glue, spaceskip)
5525             node.setglue(n, intraspace.b * quad,
5526               intraspace.p * quad,
5527               intraspace.m * quad)
5528             node.insert_before(head, item, n)
5529             node.remove(head, item)
5530           end
5531         end
5532       end
5533     end
5534   end
5535 }^^
5536 \bbl@luahyphenate}

```

10.5 CJK line breaking

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

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

```

5537 \catcode`\%=14
5538 \gdef\bbl@cjk intraspace{%
5539   \let\bbl@cjk intraspace\relax
5540   \directlua{
5541     Babel = Babel or {}
5542     require('babel-data-cjk.lua')
5543     Babel.cjk_enabled = true
5544     function Babel.cjk_linebreak(head)

```

```

5545     local GLYPH = node.id'glyph'
5546     local last_char = nil
5547     local quad = 655360      % 10 pt = 655360 = 10 * 65536
5548     local last_class = nil
5549     local last_lang = nil
5550
5551     for item in node.traverse(head) do
5552         if item.id == GLYPH then
5553
5554             local lang = item.lang
5555
5556             local LOCALE = node.get_attribute(item,
5557                 Babel.attr_locale)
5558             local props = Babel.locale_props[LOCALE]
5559
5560             local class = Babel.cjk_class[item.char].c
5561
5562             if props.cjk_quotes and props.cjk_quotes[item.char] then
5563                 class = props.cjk_quotes[item.char]
5564             end
5565
5566             if class == 'cp' then class = 'cl' end % ]] as CL
5567             if class == 'id' then class = 'I' end
5568
5569             local br = 0
5570             if class and last_class and Babel.cjk_breaks[last_class][class] then
5571                 br = Babel.cjk_breaks[last_class][class]
5572             end
5573
5574             if br == 1 and props.linebreak == 'c' and
5575                 lang ~= \the\l@nohyphenation\space and
5576                 last_lang ~= \the\l@nohyphenation then
5577                 local intrapenalty = props.intrapenalty
5578                 if intrapenalty ~= 0 then
5579                     local n = node.new(14, 0)      % penalty
5580                     n.penalty = intrapenalty
5581                     node.insert_before(head, item, n)
5582                 end
5583                 local intraspace = props.intraspace
5584                 local n = node.new(12, 13)      % (glue, spaceskip)
5585                 node.setglue(n, intraspace.b * quad,
5586                     intraspace.p * quad,
5587                     intraspace.m * quad)
5588                 node.insert_before(head, item, n)
5589             end
5590
5591             if font.getfont(item.font) then
5592                 quad = font.getfont(item.font).size
5593             end
5594             last_class = class
5595             last_lang = lang
5596             else % if penalty, glue or anything else
5597                 last_class = nil
5598             end
5599         end
5600         lang.hyphenate(head)
5601     end
5602 }%
5603 \bbl@luahyphenate}
5604 \gdef\bbl@luahyphenate{%
5605     \let\bbl@luahyphenate\relax
5606     \directlua{
5607         luatexbase.add_to_callback('hyphenate',

```

```

5608 function (head, tail)
5609   if Babel.linebreaking.before then
5610     for k, func in ipairs(Babel.linebreaking.before) do
5611       func(head)
5612     end
5613   end
5614   if Babel.cjk_enabled then
5615     Babel.cjk_linebreak(head)
5616   end
5617   lang.hyphenate(head)
5618   if Babel.linebreaking.after then
5619     for k, func in ipairs(Babel.linebreaking.after) do
5620       func(head)
5621     end
5622   end
5623   if Babel.sea_enabled then
5624     Babel.sea_disc_to_space(head)
5625   end
5626 end,
5627 'Babel.hyphenate')
5628 }
5629 }
5630 \endgroup
5631 \def\bbl@provide@intraspace{%
5632   \bbl@ifunset{\bbl@intsp@{language}}{%
5633     {\expandafter\ifx\csname\bbl@intsp@{language}\endcsname\@empty\else
5634       \bbl@xin@{c}{\bbl@cl{lnbrk}}}%
5635     \ifin@           % cjk
5636       \bbl@cjk@intraspace
5637       \directlua{
5638         Babel = Babel or {}
5639         Babel.locale_props = Babel.locale_props or {}
5640         Babel.locale_props[\the\localeid].linebreak = 'c'
5641       }%
5642       \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@{}%
5643       \ifx\bbl@KVP@intrapenalty\@nnil
5644         \bbl@intrapenalty0\@{}
5645       \fi
5646     \else           % sea
5647       \bbl@sea@intraspace
5648       \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@{}%
5649       \directlua{
5650         Babel = Babel or {}
5651         Babel.sea_ranges = Babel.sea_ranges or {}
5652         Babel.set_chranges('\bbl@cl{sbcpr}',
5653           '\bbl@cl{chrng}')
5654       }%
5655       \ifx\bbl@KVP@intrapenalty\@nnil
5656         \bbl@intrapenalty0\@{}
5657       \fi
5658     \fi
5659   \fi
5660   \ifx\bbl@KVP@intrapenalty\@nnil\else
5661     \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@{}
5662   \fi}}

```

10.6 Arabic justification

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

```

5663 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5664 \def\bblar@chars{%
5665   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%

```

```

5666 0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5667 0640,0641,0642,0643,0644,0645,0646,0647,0649}
5668 \def\bblar@elongated{%
5669 0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5670 063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5671 0649,064A}
5672 \beginingroup
5673 \catcode`_ =11 \catcode`:=11
5674 \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5675 \endgroup
5676 \gdef\bbl@arabicjust{% TODO. Allow for several locales.
5677 \let\bbl@arabicjust\relax
5678 \newattribute\bblar@kashida
5679 \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5680 \bblar@kashida=z@
5681 \bbl@patchfont{\bbl@parsejalt}}%
5682 \directlua{
5683   Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5684   Babel.arabic.elong_map[\the\localeid] = {}
5685   luatexbase.add_to_callback('post_linebreak_filter',
5686     Babel.arabic.justify, 'Babel.arabic.justify')
5687   luatexbase.add_to_callback('hpack_filter',
5688     Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5689 }%

```

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

```

5690 \def\bblar@fetchjalt#1#2#3#4{%
5691 \bbl@exp{\bbl@foreach{#1}}{%
5692 \bbl@ifunset\bblar@JE@##1}%
5693 {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"##1#2}}%
5694 {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"@nameuse\bblar@JE@##1#2}}%
5695 \directlua{%
5696   local last = nil
5697   for item in node.traverse(tex.box[0].head) do
5698     if item.id == node.id'glyph' and item.char > 0x600 and
5699     not (item.char == 0x200D) then
5700       last = item
5701     end
5702   end
5703   Babel.arabic.#3['##1#4'] = last.char
5704 }%

```

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

```

5705 \gdef\bbl@parsejalt{%
5706 \ifx\addfontfeature\undefined\else
5707 \bbl@xin@{/e}{/\bbl@cl{\lnbrk}}%
5708 \ifin@
5709 \directlua{%
5710   if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5711     Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5712     tex.print([\string\csname\space bbl@parsejalti\endcsname])
5713   end
5714 }%
5715 \fi
5716 \fi}
5717 \gdef\bbl@parsejalti{%
5718 \beginingroup
5719 \let\bbl@parsejalt\relax % To avoid infinite loop
5720 \edef\bbl@tempb{\fontid\font}%
5721 \bblar@nofswarn
5722 \bblar@fetchjalt\bblar@elongated{{from}}}%
5723 \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
5724 \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh

```

```

5725 \addfontfeature{RawFeature=+jalt}%
5726 % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5727 \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5728 \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
5729 \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
5730 \directlua{%
5731     for k, v in pairs(Babel.arabic.from) do
5732         if Babel.arabic.dest[k] and
5733             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5734             Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5735                 [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5736         end
5737     end
5738 }%
5739 \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```

5740 \begingroup
5741 \catcode`#=11
5742 \catcode`~=11
5743 \directlua{
5744
5745 Babel.arabic = Babel.arabic or {}
5746 Babel.arabic.from = {}
5747 Babel.arabic.dest = {}
5748 Babel.arabic.justify_factor = 0.95
5749 Babel.arabic.justify_enabled = true
5750 Babel.arabic.kashida_limit = -1
5751
5752 function Babel.arabic.justify(head)
5753     if not Babel.arabic.justify_enabled then return head end
5754     for line in node.traverse_id(node.id'hlist', head) do
5755         Babel.arabic.justify_hlist(head, line)
5756     end
5757     return head
5758 end
5759
5760 function Babel.arabic.justify_hbox(head, gc, size, pack)
5761     local has_inf = false
5762     if Babel.arabic.justify_enabled and pack == 'exactly' then
5763         for n in node.traverse_id(12, head) do
5764             if n.stretch_order > 0 then has_inf = true end
5765         end
5766         if not has_inf then
5767             Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5768         end
5769     end
5770     return head
5771 end
5772
5773 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5774     local d, new
5775     local k_list, k_item, pos_inline
5776     local width, width_new, full, k_curr, wt_pos, goal, shift
5777     local subst_done = false
5778     local elong_map = Babel.arabic.elong_map
5779     local cnt
5780     local last_line
5781     local GLYPH = node.id'glyph'
5782     local KASHIDA = Babel.attr_kashida
5783     local LOCALE = Babel.attr_locale
5784
5785     if line == nil then

```



```

5786     line = {}
5787     line.glue_sign = 1
5788     line.glue_order = 0
5789     line.head = head
5790     line.shift = 0
5791     line.width = size
5792 end
5793
5794 % Exclude last line. todo. But-- it discards one-word lines, too!
5795 % ? Look for glue = 12:15
5796 if (line.glue_sign == 1 and line.glue_order == 0) then
5797     elongs = {}      % Stores elongated candidates of each line
5798     k_list = {}      % And all letters with kashida
5799     pos_inline = 0   % Not yet used
5800
5801     for n in node.traverse_id(GLYPH, line.head) do
5802         pos_inline = pos_inline + 1 % To find where it is. Not used.
5803
5804         % Elongated glyphs
5805         if elong_map then
5806             local locale = node.get_attribute(n, LOCALE)
5807             if elong_map[locale] and elong_map[locale][n.font] and
5808                 elong_map[locale][n.font][n.char] then
5809                 table.insert(elongs, {node = n, locale = locale} )
5810                 node.set_attribute(n.prev, KASHIDA, 0)
5811             end
5812         end
5813
5814         % Tatwil
5815         if Babel.kashida_wts then
5816             local k_wt = node.get_attribute(n, KASHIDA)
5817             if k_wt > 0 then % todo. parameter for multi inserts
5818                 table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5819             end
5820         end
5821
5822     end % of node.traverse_id
5823
5824     if #elongs == 0 and #k_list == 0 then goto next_line end
5825     full = line.width
5826     shift = line.shift
5827     goal = full * Babel.arabic.justify_factor % A bit crude
5828     width = node.dimensions(line.head) % The 'natural' width
5829
5830     % == Elongated ==
5831     % Original idea taken from 'chickenize'
5832     while (#elongs > 0 and width < goal) do
5833         subst_done = true
5834         local x = #elongs
5835         local curr = elongs[x].node
5836         local oldchar = curr.char
5837         curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5838         width = node.dimensions(line.head) % Check if the line is too wide
5839         % Substitute back if the line would be too wide and break:
5840         if width > goal then
5841             curr.char = oldchar
5842             break
5843         end
5844         % If continue, pop the just substituted node from the list:
5845         table.remove(elongs, x)
5846     end
5847
5848     % == Tatwil ==

```

```

5849   if #k_list == 0 then goto next_line end
5850
5851   width = node.dimensions(line.head)    % The 'natural' width
5852   k_curr = #k_list % Traverse backwards, from the end
5853   wt_pos = 1
5854
5855   while width < goal do
5856     subst_done = true
5857     k_item = k_list[k_curr].node
5858     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5859       d = node.copy(k_item)
5860       d.char = 0x0640
5861       d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
5862       d.xoffset = 0
5863       line.head, new = node.insert_after(line.head, k_item, d)
5864       width_new = node.dimensions(line.head)
5865       if width > goal or width == width_new then
5866         node.remove(line.head, new) % Better compute before
5867         break
5868       end
5869       if Babel.fix_diacr then
5870         Babel.fix_diacr(k_item.next)
5871       end
5872       width = width_new
5873     end
5874     if k_curr == 1 then
5875       k_curr = #k_list
5876       wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5877     else
5878       k_curr = k_curr - 1
5879     end
5880   end
5881
5882   % Limit the number of tatweel by removing them. Not very efficient,
5883   % but it does the job in a quite predictable way.
5884   if Babel.arabic.kashida_limit > -1 then
5885     cnt = 0
5886     for n in node.traverse_id(GLYPH, line.head) do
5887       if n.char == 0x0640 then
5888         cnt = cnt + 1
5889         if cnt > Babel.arabic.kashida_limit then
5890           node.remove(line.head, n)
5891         end
5892       else
5893         cnt = 0
5894       end
5895     end
5896   end
5897
5898   ::next_line::
5899
5900   % Must take into account marks and ins, see luatex manual.
5901   % Have to be executed only if there are changes. Investigate
5902   % what's going on exactly.
5903   if subst_done and not gc then
5904     d = node.hpack(line.head, full, 'exactly')
5905     d.shift = shift
5906     node.insert_before(head, line, d)
5907     node.remove(head, line)
5908   end
5909 end % if process line
5910 end
5911 }

```

```

5912 \endgroup
5913 \fi\fi % ends Arabic just block: \ifnum\bbl@bidimode>100...

```

10.7 Common stuff

```

5914 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5915 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5916 \DisableBabelHook{babel-fontspec}
5917 <<Font selection>>

```

10.8 Automatic fonts and ids switching

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

```

5918% TODO - to a lua file
5919 \directlua{
5920 Babel.script_blocks = {
5921   ['dflt'] = {},
5922   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5923               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5924   ['Armn'] = {{0x0530, 0x058F}},
5925   ['Beng'] = {{0x0980, 0x09FF}},
5926   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0ABBF}},
5927   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5928   ['Cyril'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5929               {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5930   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5931   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5932               {0xAB00, 0xAB2F}},
5933   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5934   % Don't follow strictly Unicode, which places some Coptic letters in
5935   % the 'Greek and Coptic' block
5936   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5937   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5938               {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5939               {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5940               {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5941               {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5942               {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5943   ['Hebr'] = {{0x0590, 0x05FF}},
5944   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5945               {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5946   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5947   ['Knda'] = {{0x0C80, 0x0CFF}},
5948   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5949               {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5950               {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5951   ['Laoo'] = {{0x0E80, 0x0EFF}},
5952   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5953               {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5954               {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5955   ['Mahj'] = {{0x11150, 0x1117F}},
5956   ['Mlym'] = {{0x0D00, 0x0D7F}},
5957   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5958   ['Orya'] = {{0x0B00, 0x0B7F}},
5959   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5960   ['Syrn'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},

```

```

5961 ['Taml'] = {{0x0B80, 0x0BFF}},
5962 ['Telu'] = {{0x0C00, 0x0C7F}},
5963 ['Tfng'] = {{0x2D30, 0x2D7F}},
5964 ['Thai'] = {{0x0E00, 0x0E7F}},
5965 ['Tibt'] = {{0x0F00, 0x0FFF}},
5966 ['Vaii'] = {{0xA500, 0xA63F}},
5967 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5968 }
5969
5970 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5971 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5972 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5973
5974 function Babel.locale_map(head)
5975   if not Babel.locale_mapped then return head end
5976
5977   local LOCALE = Babel.attr_locale
5978   local GLYPH = node.id('glyph')
5979   local inmath = false
5980   local toloc_save
5981   for item in node.traverse(head) do
5982     local toloc
5983     if not inmath and item.id == GLYPH then
5984       % Optimization: build a table with the chars found
5985       if Babel.chr_to_loc[item.char] then
5986         toloc = Babel.chr_to_loc[item.char]
5987       else
5988         for lc, maps in pairs(Babel.loc_to_scr) do
5989           for _, rg in pairs(maps) do
5990             if item.char >= rg[1] and item.char <= rg[2] then
5991               Babel.chr_to_loc[item.char] = lc
5992               toloc = lc
5993               break
5994             end
5995           end
5996         end
5997         % Treat composite chars in a different fashion, because they
5998         % 'inherit' the previous locale.
5999         if (item.char >= 0x0300 and item.char <= 0x036F) or
6000            (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
6001            (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
6002           Babel.chr_to_loc[item.char] = -2000
6003           toloc = -2000
6004         end
6005         if not toloc then
6006           Babel.chr_to_loc[item.char] = -1000
6007         end
6008       end
6009       if toloc == -2000 then
6010         toloc = toloc_save
6011       elseif toloc == -1000 then
6012         toloc = nil
6013       end
6014       if toloc and Babel.locale_props[toloc] and
6015          Babel.locale_props[toloc].letters and
6016          tex.getcatcode(item.char) \string~= 11 then
6017         toloc = nil
6018       end
6019       if toloc and Babel.locale_props[toloc].script
6020          and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6021          and Babel.locale_props[toloc].script ==
6022          Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6023         toloc = nil

```

```

6024     end
6025     if toloc then
6026         if Babel.locale_props[toloc].lg then
6027             item.lang = Babel.locale_props[toloc].lg
6028             node.set_attribute(item, LOCALE, toloc)
6029         end
6030         if Babel.locale_props[toloc]['/'..item.font] then
6031             item.font = Babel.locale_props[toloc]['/'..item.font]
6032         end
6033     end
6034     toloc_save = toloc
6035     elseif not inmath and item.id == 7 then % Apply recursively
6036         item.replace = item.replace and Babel.locale_map(item.replace)
6037         item.pre      = item.pre and Babel.locale_map(item.pre)
6038         item.post      = item.post and Babel.locale_map(item.post)
6039     elseif item.id == node.id'math' then
6040         inmath = (item.subtype == 0)
6041     end
6042 end
6043 return head
6044 end
6045 }

```

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

```

6046 \newcommand\babelcharproperty[1]{%
6047   \count@=#1\relax
6048   \ifvmode
6049     \expandafter\bbl@chprop
6050   \else
6051     \bbl@error{\string\babelcharproperty\space can be used only in\%
6052               vertical mode (preamble or between paragraphs)}%
6053     {See the manual for further info}%
6054   \fi}
6055 \newcommand\bbl@chprop[3][\the\count@]{%
6056   \@tempcnta=#1\relax
6057   \bbl@ifunset{\bbl@chprop@#2}%
6058   {\bbl@error{No property named '#2'. Allowed values are\%
6059             direction (bc), mirror (bmg), and linebreak (lb)}%
6060    {See the manual for further info}}%
6061   {%
6062   \loop
6063     \bbl@cs{chprop@#2}{#3}%
6064     \ifnum\count@<\@tempcnta
6065       \advance\count@\@ne
6066     \repeat}
6067 \def\bbl@chprop@direction#1{%
6068   \directlua{
6069     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6070     Babel.characters[\the\count@]['d'] = '#1'
6071   }}
6072 \let\bbl@chprop@bc\bbl@chprop@direction
6073 \def\bbl@chprop@mirror#1{%
6074   \directlua{
6075     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6076     Babel.characters[\the\count@]['m'] = '\number#1'
6077   }}
6078 \let\bbl@chprop@bmg\bbl@chprop@mirror
6079 \def\bbl@chprop@linebreak#1{%
6080   \directlua{
6081     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6082     Babel.cjk_characters[\the\count@]['c'] = '#1'
6083   }}

```

```

6084 \let\bbl@chprop@lb\bbl@chprop@linebreak
6085 \def\bbl@chprop@locale#1{%
6086   \directlua{
6087     Babel.chr_to_loc = Babel.chr_to_loc or {}
6088     Babel.chr_to_loc[\the\count@] =
6089       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
6090   }}

```

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.

```

6091 \directlua{
6092   Babel.nohyphenation = \the\l@nohyphenation
6093 }

```

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).

```

6094 \begingroup
6095 \catcode`\~ = 12
6096 \catcode`\% = 12
6097 \catcode`\& = 14
6098 \catcode`\| = 12
6099 \gdef\babelprehyphenation{%&
6100   \@ifnextchar[\bbl@settransform{0}]{\bbl@settransform{0}[]}}
6101 \gdef\babelposthyphenation{%&
6102   \@ifnextchar[\bbl@settransform{1}]{\bbl@settransform{1}[]}}
6103 \gdef\bbl@settransform#1[#2]#3#4#5{%&
6104   \ifcase#1
6105     \bbl@activateprehyphen
6106   \or
6107     \bbl@activateposthyphen
6108   \fi
6109   \begingroup
6110     \def\babeltempa{\bbl@add@list\babeltempb}%&
6111     \let\babeltempb@empty
6112     \def\bbl@tempa{#5}%&
6113     \bbl@replace\bbl@tempa{,}{ ,}%& TODO. Ugly trick to preserve {}
6114     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{%&
6115       \bbl@ifsamestring{##1}{remove}%&
6116       {\bbl@add@list\babeltempb{nil}}}%&
6117     {\directlua{
6118       local rep = {[#1]}
6119       rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6120       rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
6121       rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
6122       if #1 == 0 or #1 == 2 then
6123         rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
6124           'space = { ' .. '%2, %3, %4' .. ' }')
6125         rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
6126           'spacefactor = { ' .. '%2, %3, %4' .. ' }')
6127         rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
6128       else
6129         rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
6130         rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
6131         rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
6132       end
6133       tex.print([[string\babeltempa{[]] .. rep .. [{}]])
6134     ]}}%&

```

```

6135 \bbl@foreach\babeltempb{%&
6136 \bbl@forkv{##1}}{%&
6137 \in{,###1,},{,nil,step,data,remove,insert,string,no,pre,&
6138 no,post,penalty,kashida,space,spacefactor,}%&
6139 \ifin\else
6140 \bbl@error
6141 {Bad option '###1' in a transform.\\&
6142 I'll ignore it but expect more errors}%&
6143 {See the manual for further info.}%&
6144 \fi}}{%&
6145 \let\bbl@kv@attribute\relax
6146 \let\bbl@kv@label\relax
6147 \let\bbl@kv@fonts\@empty
6148 \bbl@forkv{#2}{\bbl@csarg\edef{kv##1}{##2}}{%&
6149 \ifx\bbl@kv@fonts\@empty\else\bbl@settransfont\fi
6150 \ifx\bbl@kv@attribute\relax
6151 \ifx\bbl@kv@label\relax\else
6152 \bbl@exp{\\bbl@trim\def\\bbl@kv@fonts{\bbl@kv@fonts}}{%&
6153 \bbl@replace\bbl@kv@fonts{ },}%&
6154 \edef\bbl@kv@attribute{\bbl@ATR@\bbl@kv@label @#3@\bbl@kv@fonts}%&
6155 \count@ \z@
6156 \def\bbl@elt##1##2##3{%&
6157 \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}%&
6158 {\bbl@ifsamestring{\bbl@kv@fonts}{##3}%&
6159 {\count@\@ne}%&
6160 {\bbl@error
6161 {Transforms cannot be re-assigned to different\\&
6162 fonts. The conflict is in '\bbl@kv@label'.\\&
6163 Apply the same fonts or use a different label}%&
6164 {See the manual for further details.}}}%&
6165 {}}}{%&
6166 \bbl@transfont@list
6167 \ifnum\count@=\z@
6168 \bbl@exp{\global\\bbl@add\\bbl@transfont@list
6169 {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}%&
6170 \fi
6171 \bbl@ifunset{\bbl@kv@attribute}%&
6172 {\global\bbl@carg\newattribute{\bbl@kv@attribute}}{%&
6173 }{%&
6174 \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
6175 \fi
6176 \else
6177 \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}%&
6178 \fi
6179 \directlua{
6180 local lbkr = Babel.linebreaking.replacements[#1]
6181 local u = unicode.utf8
6182 local id, attr, label
6183 if #1 == 0 then
6184 id = \the\csname bbl@id@#3\endcsname\space
6185 else
6186 id = \the\csname l@#3\endcsname\space
6187 end
6188 \ifx\bbl@kv@attribute\relax
6189 attr = -1
6190 \else
6191 attr = luatexbase.registernumber'\bbl@kv@attribute'
6192 \fi
6193 \ifx\bbl@kv@label\relax\else & Same refs:
6194 label = {[=[\bbl@kv@label]=]}
6195 \fi
6196 & Convert pattern:
6197 local patt = string.gsub([=[#4]=], '%s', '')

```

```

6198     if #1 == 0 then
6199         patt = string.gsub(patt, '|', ' ')
6200     end
6201     if not u.find(patt, '()', nil, true) then
6202         patt = '()' .. patt .. '()'
6203     end
6204     if #1 == 1 then
6205         patt = string.gsub(patt, '%(%)^', '^()')
6206         patt = string.gsub(patt, '%$%(%)', '()$')
6207     end
6208     patt = u.gsub(patt, '{(.)}',
6209         function (n)
6210             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6211         end)
6212     patt = u.gsub(patt, '{(%x%x%x%x+)}',
6213         function (n)
6214             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6215         end)
6216     lbkr[id] = lbkr[id] or {}
6217     table.insert(lbkr[id],
6218         { label=label, attr=attr, pattern=patt, replace={\blabeltempb} })
6219 }&%
6220 \endgroup}
6221 \endgroup
6222 \let\bbl@transfont@list@empty
6223 \def\bbl@settransfont{%
6224     \global\let\bbl@settransfont\relax % Execute only once
6225     \gdef\bbl@transfont{%
6226         \def\bbl@elt####1####2####3{%
6227             \bbl@ifblank{####3}%
6228             {\count@tw@}% Do nothing if no fonts
6229             {\count@z@
6230             \bbl@vforeach{####3}{%
6231                 \def\bbl@tempd{#####1}%
6232                 \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6233                 \ifx\bbl@tempd\bbl@tempe
6234                     \count@\@ne
6235                 \else\ifx\bbl@tempd\bbl@transfam
6236                     \count@\@ne
6237                 \fi\fi}%
6238             \ifcase\count@
6239                 \bbl@csarg\unsetattribute{ATR####2@####1@####3}%
6240             \or
6241                 \bbl@csarg\setattribute{ATR####2@####1@####3}\@ne
6242             \fi}}%
6243         \bbl@transfont@list}%
6244     \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6245     \gdef\bbl@transfam{-unknown-}%
6246     \bbl@foreach\bbl@font@fams{%
6247         \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6248         \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6249         {\xdef\bbl@transfam{##1}}%
6250     {}}}
6251 \DeclareRobustCommand\enablelocaletransform[1]{%
6252     \bbl@ifunset{\bbl@ATR#1@\languagename @}%
6253     {\bbl@error
6254         {'#1' for '\languagename' cannot be enabled.\%
6255         Maybe there is a typo or it's a font-dependent transform}%
6256         {See the manual for further details.}}%
6257     {\bbl@csarg\setattribute{ATR#1@\languagename @}\@ne}}
6258 \DeclareRobustCommand\disablelocaletransform[1]{%
6259     \bbl@ifunset{\bbl@ATR#1@\languagename @}%
6260     {\bbl@error

```



```

6261      {'#1' for '\language' cannot be disabled.\\%
6262      Maybe there is a typo or it's a font-dependent transform}%
6263      {See the manual for further details.}}%
6264      {\bbl@csarg\unsetAttribute{ATR@#1@\language @}}
6265 \def\bbl@activateposthyphen{%
6266   \let\bbl@activateposthyphen\relax
6267   \directlua{
6268     require('babel-transforms.lua')
6269     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6270   }}
6271 \def\bbl@activateprehyphen{%
6272   \let\bbl@activateprehyphen\relax
6273   \directlua{
6274     require('babel-transforms.lua')
6275     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6276   }}

```

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.

```

6277 \newcommand\localeprehyphenation[1]{%
6278   \directlua{ Babel.string_prehyphenation([=#1]=], \the\localeid) }}

```

10.9 Bidi

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

```

6279 \def\bbl@activate@preotf{%
6280   \let\bbl@activate@preotf\relax % only once
6281   \directlua{
6282     Babel = Babel or {}
6283     %
6284     function Babel.pre_otfload_v(head)
6285       if Babel.numbers and Babel.digits_mapped then
6286         head = Babel.numbers(head)
6287       end
6288       if Babel.bidi_enabled then
6289         head = Babel.bidi(head, false, dir)
6290       end
6291       return head
6292     end
6293     %
6294     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6295       if Babel.numbers and Babel.digits_mapped then
6296         head = Babel.numbers(head)
6297       end
6298       if Babel.bidi_enabled then
6299         head = Babel.bidi(head, false, dir)
6300       end
6301       return head
6302     end
6303     %
6304     luatexbase.add_to_callback('pre_linebreak_filter',
6305       Babel.pre_otfload_v,
6306       'Babel.pre_otfload_v',
6307     luatexbase.priority_in_callback('pre_linebreak_filter',
6308       'luaotfload.node_processor') or nil)
6309     %
6310     luatexbase.add_to_callback('hpack_filter',
6311       Babel.pre_otfload_h,

```

```

6312     'Babel.pre_otfload_h',
6313     luatexbase.priority_in_callback('hpack_filter',
6314     'luaotfload.node_processor') or nil)
6315 }}

```

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=`.

```

6316 \breakafterdirmode=1
6317 \ifnum\bbl@bidimode>\@ne % Any bidi= except default=1
6318   \let\bbl@beforeforeign\leavevmode
6319   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6320   \RequirePackage{luatexbase}
6321   \bbl@activate@preotf
6322   \directlua{
6323     require('babel-data-bidi.lua')
6324     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6325       require('babel-bidi-basic.lua')
6326     \or
6327       require('babel-bidi-basic-r.lua')
6328     \fi}
6329   \newattribute\bbl@attr@dir
6330   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6331   \bbl@exp{\output{\bodydir\pagedir\the\output}}
6332 \fi
6333 \chardef\bbl@thetextdir\z@
6334 \chardef\bbl@thepardir\z@
6335 \def\bbl@getluadir#1{%
6336   \directlua{
6337     if tex.#ldir == 'TLT' then
6338       tex.sprint('0')
6339     elseif tex.#ldir == 'TRT' then
6340       tex.sprint('1')
6341     end}}
6342 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6343   \ifcase#3\relax
6344     \ifcase\bbl@getluadir{#1}\relax\else
6345       #2 TLT\relax
6346     \fi
6347   \else
6348     \ifcase\bbl@getluadir{#1}\relax
6349       #2 TRT\relax
6350     \fi
6351   \fi}
6352 % ..00PPTT, with masks 0xC (par dir) and 0x3 (text dir)
6353 \def\bbl@thedir{0}
6354 \def\bbl@textdir#1{%
6355   \bbl@setluadir{text}\textdir{#1}%
6356   \chardef\bbl@thetextdir#1\relax
6357   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6358   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6359 \def\bbl@pardir#1{% Used twice
6360   \bbl@setluadir{par}\pardir{#1}%
6361   \chardef\bbl@thepardir#1\relax}
6362 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}% Used once
6363 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}% Unused
6364 \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.

```

6365 \ifnum\bbl@bidimode>\z@ % Any bidi=
6366   \def\bbl@insidemath{0}%
6367   \def\bbl@everymath{\def\bbl@insidemath{1}}
6368   \def\bbl@everydisplay{\def\bbl@insidemath{2}}

```

```

6369 \frozen@everymath\expandafter{%
6370   \expandafter\bbl@everymath\the\frozen@everymath}
6371 \frozen@everydisplay\expandafter{%
6372   \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6373 \AtBeginDocument{
6374   \directlua{
6375     function Babel.math_box_dir(head)
6376       if not (token.get_macro('bbl@insidemath') == '0') then
6377         if Babel.hlist_has_bidi(head) then
6378           local d = node.new(node.id'dir')
6379           d.dir = '+TRT'
6380           node.insert_before(head, node.has_glyph(head), d)
6381           for item in node.traverse(head) do
6382             node.set_attribute(item,
6383               Babel.attr_dir, token.get_macro('bbl@thedir'))
6384           end
6385         end
6386       end
6387       return head
6388     end
6389     luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6390       "Babel.math_box_dir", 0)
6391   }}%
6392 \fi

```

10.10 Layout

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

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

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

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

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

```

6393 \bbl@trace{Redefinitions for bidi layout}
6394 %
6395 <<(*More package options)>> ≡
6396 \chardef\bbl@eqnpos\z@
6397 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6398 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6399 <</More package options>>
6400 %
6401 \ifnum\bbl@bidimode>\z@ % Any bidi=
6402   \matheqdirmode\@ne % A luatex primitive
6403   \let\bbl@eqnodir\relax
6404   \def\bbl@eqdel{()}
6405   \def\bbl@eqnum{%
6406     {\normalfont\normalcolor
6407       \expandafter\@firstoftwo\bbl@eqdel
6408       \theequation
6409       \expandafter\@secondoftwo\bbl@eqdel}}

```

```

6410 \def\bbl@puteqno#1{\eqno\hbox{#1}}
6411 \def\bbl@putleqno#1{\leqno\hbox{#1}}
6412 \def\bbl@eqno@flip#1{%
6413   \ifdim\predisplaysize=-\maxdimen
6414     \eqno
6415     \hb@xt@.01pt{%
6416       \hb@xt@\displaywidth{\hss{#1\glet\bbl@upset\@currentlabel}}\hss}%
6417   \else
6418     \leqno\hbox{#1\glet\bbl@upset\@currentlabel}%
6419   \fi
6420   \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6421 \def\bbl@leqno@flip#1{%
6422   \ifdim\predisplaysize=-\maxdimen
6423     \leqno
6424     \hb@xt@.01pt{%
6425       \hss\hb@xt@\displaywidth{\hss{#1\glet\bbl@upset\@currentlabel}}\hss}%
6426   \else
6427     \eqno\hbox{#1\glet\bbl@upset\@currentlabel}%
6428   \fi
6429   \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6430 \AtBeginDocument{%
6431   \ifx\bbl@noamsmath\relax\else
6432   \ifx\maketag@@@undefined % Normal equation, eqnarray
6433     \AddToHook{env/equation/begin}{%
6434       \ifnum\bbl@thetextdir>\z@
6435         \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6436         \let\@eqnnum\bbl@eqnum
6437         \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6438         \chardef\bbl@thetextdir\z@
6439         \bbl@add\normalfont{\bbl@eqnodir}%
6440         \ifcase\bbl@eqnpos
6441           \let\bbl@puteqno\bbl@eqno@flip
6442         \or
6443           \let\bbl@puteqno\bbl@leqno@flip
6444         \fi
6445       \fi}%
6446   \ifnum\bbl@eqnpos=\tw@ \else
6447     \def\endequation{\bbl@puteqno{\@eqnnum}\$@ignoretrue}%
6448   \fi
6449   \AddToHook{env/eqnarray/begin}{%
6450     \ifnum\bbl@thetextdir>\z@
6451       \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6452       \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6453       \chardef\bbl@thetextdir\z@
6454       \bbl@add\normalfont{\bbl@eqnodir}%
6455       \ifnum\bbl@eqnpos=\@ne
6456         \def\@eqnnum{%
6457           \setbox\z@\hbox{\bbl@eqnum}%
6458           \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6459       \else
6460         \let\@eqnnum\bbl@eqnum
6461       \fi
6462     \fi}
6463   % Hack. YA luatex bug?:
6464   \expandafter\bbl@sreplace\csname] \endcsname{\$}{\eqno\kern.001pt\$}%
6465 \else % amstex
6466   \bbl@exp{% Hack to hide maybe undefined conditionals:
6467     \chardef\bbl@eqnpos=0%
6468     \<iftagsleft>1\<else>\<if\leqn>2\<fi>\<fi>\relax}%
6469   \ifnum\bbl@eqnpos=\@ne
6470     \let\bbl@ams@lap\hbox
6471   \else
6472     \let\bbl@ams@lap\llap

```

```

6473 \fi
6474 \ExplSyntaxOn % Required by \bbl@sreplace with \intertext@
6475 \bbl@sreplace\intertext@\normalbaselines}%
6476 {\normalbaselines
6477 \ifx\bbl@eqnodir\relax\else\bbl@pardir\@ne\bbl@eqnodir\fi}%
6478 \ExplSyntaxOff
6479 \def\bbl@ams@tagbox#1#2{#1{\bbl@eqnodir#2}}% #1=hbox|@lap|flip
6480 \ifx\bbl@ams@lap\hbox % leqno
6481 \def\bbl@ams@flip#1{%
6482 \hbox to 0.01pt{\hss\hbox to\displaywidth{#1}\hss}}%
6483 \else % eqno
6484 \def\bbl@ams@flip#1{%
6485 \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}}\hss}}%
6486 \fi
6487 \def\bbl@ams@preset#1{%
6488 \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6489 \ifnum\bbl@thetextdir>\z@
6490 \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6491 \bbl@sreplace\textdef@{\hbox}{\bbl@ams@tagbox\hbox}%
6492 \bbl@sreplace\maketag@@@{\hbox}{\bbl@ams@tagbox#1}%
6493 \fi}%
6494 \ifnum\bbl@eqnpos=\tw@ \else
6495 \def\bbl@ams@equation{%
6496 \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6497 \ifnum\bbl@thetextdir>\z@
6498 \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6499 \chardef\bbl@thetextdir\z@
6500 \bbl@add\normalfont{\bbl@eqnodir}%
6501 \ifcase\bbl@eqnpos
6502 \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
6503 \or
6504 \def\veqno##1##2{\bbl@leqno@flip{##1##2}}%
6505 \fi
6506 \fi}%
6507 \AddToHook{env/equation/begin}{\bbl@ams@equation}%
6508 \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6509 \fi
6510 \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6511 \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6512 \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6513 \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6514 \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6515 \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6516 \AddToHook{env/alignat/begin}{\bbl@ams@preset\bbl@ams@lap}%
6517 \AddToHook{env/alignat*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6518 \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6519 % Hackish, for proper alignment. Don't ask me why it works!:
6520 \bbl@exp% Avoid a 'visible' conditional
6521 \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\tag*{\<fi>}}%
6522 \\\AddToHook{env/alignat*/end}{\<iftag@>\<else>\\tag*{\<fi>}}%
6523 \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6524 \AddToHook{env/split/before}{%
6525 \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6526 \ifnum\bbl@thetextdir>\z@
6527 \bbl@ifsamestring\@currentenv{equation}%
6528 {\ifx\bbl@ams@lap\hbox % leqno
6529 \def\bbl@ams@flip#1{%
6530 \hbox to 0.01pt{\hbox to\displaywidth{#1}\hss}\hss}}%
6531 \else
6532 \def\bbl@ams@flip#1{%
6533 \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}%
6534 \fi}%
6535 {}%

```

```

6536         \fi}%
6537     \fi\fi}
6538 \fi
6539 \def\bbl@provide@extra#1{%
6540     % == Counters: mapdigits ==
6541     % Native digits
6542     \ifx\bbl@KVP@mapdigits\@nnil\else
6543         \bbl@ifunset\bbl@dgnat@\language\name{}{}%
6544         {\RequirePackage{luatexbase}%
6545         \bbl@activate@preotf
6546         \directlua{
6547             Babel = Babel or {} %%% -> presets in luababel
6548             Babel.digits_mapped = true
6549             Babel.digits = Babel.digits or {}
6550             Babel.digits[\the\localeid] =
6551             table.pack(string.utfvalue('\bbl@cl{dgnat}'))
6552             if not Babel.numbers then
6553                 function Babel.numbers(head)
6554                     local LOCALE = Babel.attr_locale
6555                     local GLYPH = node.id'glyph'
6556                     local inmath = false
6557                     for item in node.traverse(head) do
6558                         if not inmath and item.id == GLYPH then
6559                             local temp = node.get_attribute(item, LOCALE)
6560                             if Babel.digits[temp] then
6561                                 local chr = item.char
6562                                 if chr > 47 and chr < 58 then
6563                                     item.char = Babel.digits[temp][chr-47]
6564                                 end
6565                             end
6566                             elseif item.id == node.id'math' then
6567                                 inmath = (item.subtype == 0)
6568                             end
6569                         end
6570                     end
6571                     return head
6572                 end
6573             }}%
6574         \fi
6575     % == transforms ==
6576     \ifx\bbl@KVP@transforms\@nnil\else
6577         \def\bbl@elt##1##2##3{%
6578             \in@{$transforms.}{$##1}%
6579             \ifin@
6580                 \def\bbl@tempa{##1}%
6581                 \bbl@replace\bbl@tempa{transforms.}{}%
6582                 \bbl@carg\bbl@transforms{babel\bbl@tempa}{##2}{##3}%
6583             \fi}%
6584         \csname bbl@inidata@\language\name\endcsname
6585         \bbl@release@transforms\relax % \relax closes the last item.
6586     \fi}
6587 % Start tabular here:
6588 \def\localerestoredirs{%
6589     \ifcase\bbl@thetextdir
6590         \ifnum\textdirection=\z@\else\textdir TLT\fi
6591     \else
6592         \ifnum\textdirection=\@ne\else\textdir TRT\fi
6593     \fi
6594     \ifcase\bbl@thepardir
6595         \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6596     \else
6597         \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6598     \fi}

```

```

6599 \IfBabelLayout{tabular}%
6600   {\chardef\bbl@tabular@mode\tw@}% All RTL
6601   {\IfBabelLayout{notabular}%
6602     {\chardef\bbl@tabular@mode\z@}%
6603     {\chardef\bbl@tabular@mode@ne}% Mixed, with LTR cols
6604 \ifnum\bbl@bidimode>\@ne % Any lua bidi= except default=1
6605   \ifcase\bbl@tabular@mode\or % 1
6606     \let\bbl@parabefore\relax
6607     \AddToHook{para/before}{\bbl@parabefore}
6608     \AtBeginDocument{%
6609       \bbl@replace\@tabular{$}{$%
6610         \def\bbl@insidemath{0}%
6611         \def\bbl@parabefore{\localerestoredirs}}%
6612       \ifnum\bbl@tabular@mode=\@ne
6613         \bbl@ifunset{@tabclassz}{}%
6614         \bbl@exp{% Hide conditionals
6615           \\\bbl@sreplace\\ \@tabclassz
6616             {\<ifcase>\\ \@chnum}%
6617             {\\\localerestoredirs\<ifcase>\\ \@chnum}}}%
6618         \@ifpackageloaded{colortbl}%
6619           {\bbl@sreplace\@classz
6620             {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6621           {\@ifpackageloaded{array}%
6622             {\bbl@exp{% Hide conditionals
6623               \\\bbl@sreplace\\ \@classz
6624                 {\<ifcase>\\ \@chnum}%
6625                 {\bgroup\\ \localerestoredirs\<ifcase>\\ \@chnum}%
6626                 \\\bbl@sreplace\\ \@classz
6627                 {\\\do@row@strut\<fi>}{\\do@row@strut\<fi>\egroup}}}%
6628             {}}%
6629       \fi}%
6630   \or % 2
6631     \let\bbl@parabefore\relax
6632     \AddToHook{para/before}{\bbl@parabefore}%
6633     \AtBeginDocument{%
6634       \@ifpackageloaded{colortbl}%
6635         {\bbl@replace\@tabular{$}{$%
6636           \def\bbl@insidemath{0}%
6637           \def\bbl@parabefore{\localerestoredirs}}%
6638         \bbl@sreplace\@classz
6639         {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6640       {}}%
6641   \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.

```

6642 \AtBeginDocument{%
6643   \@ifpackageloaded{multicol}%
6644     {\toks@ \expandafter{\multi@column@out}%
6645       \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
6646     {}%
6647   \@ifpackageloaded{paracol}%
6648     {\edef\pcol@output{%
6649       \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
6650     {}%
6651 \fi
6652 \ifx\bbl@opt@layout@nnil\endinput\fi % if no layout

```

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

```

6653 \ifnum\bbl@bidimode>\z@ % Any bidi=

```

```

6654 \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6655 \bbl@exp{%
6656 \def\\bbl@insidemath{0}%
6657 \mathdir\the\bodydir
6658 #1% Once entered in math, set boxes to restore values
6659 \<ifmmode>%
6660 \everyvbox{%
6661 \the\everyvbox
6662 \bodydir\the\bodydir
6663 \mathdir\the\mathdir
6664 \everyhbox{\the\everyhbox}%
6665 \everyvbox{\the\everyvbox}}%
6666 \everyhbox{%
6667 \the\everyhbox
6668 \bodydir\the\bodydir
6669 \mathdir\the\mathdir
6670 \everyhbox{\the\everyhbox}%
6671 \everyvbox{\the\everyvbox}}%
6672 \<fi>}}%
6673 \def\@hangfrom#1{%
6674 \setbox\@tempboxa\hbox{{#1}}%
6675 \hangindent\wd\@tempboxa
6676 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6677 \shapemode\@ne
6678 \fi
6679 \noindent\box\@tempboxa}
6680 \fi
6681 \IfBabelLayout{tabular}
6682 {\let\bbl@OL@tabular\@tabular
6683 \bbl@replace\@tabular{$}\{\bbl@nextfake$\}%
6684 \let\bbl@NL@tabular\@tabular
6685 \AtBeginDocument{%
6686 \ifx\bbl@NL@tabular\@tabular\else
6687 \bbl@exp{\in{\bbl@nextfake}\{[@tabular]\}}%
6688 \ifin\else
6689 \bbl@replace\@tabular{$}\{\bbl@nextfake$\}%
6690 \fi
6691 \let\bbl@NL@tabular\@tabular
6692 \fi}}
6693 {}
6694 \IfBabelLayout{lists}
6695 {\let\bbl@OL@list\list
6696 \bbl@sreplace\list{\parshape}\{\bbl@listparshape}%
6697 \let\bbl@NL@list\list
6698 \def\bbl@listparshape#1#2#3{%
6699 \parshape #1 #2 #3 %
6700 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6701 \shapemode\tw@
6702 \fi}}
6703 {}
6704 \IfBabelLayout{graphics}
6705 {\let\bbl@pictresetdir\relax
6706 \def\bbl@pictsetdir#1{%
6707 \ifcase\bbl@thetextdir
6708 \let\bbl@pictresetdir\relax
6709 \else
6710 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6711 \or\textdir TLT
6712 \else\bodydir TLT \textdir TLT
6713 \fi
6714 % \(\text|par)dir required in pgf:
6715 \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6716 \fi}%

```



```

6717 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw}%
6718 \directlua{
6719   Babel.get_picture_dir = true
6720   Babel.picture_has_bidi = 0
6721   %
6722   function Babel.picture_dir (head)
6723     if not Babel.get_picture_dir then return head end
6724     if Babel.hlist_has_bidi(head) then
6725       Babel.picture_has_bidi = 1
6726     end
6727     return head
6728   end
6729   luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6730     "Babel.picture_dir")
6731 }%
6732 \AtBeginDocument{%
6733   \def\LS@rot{%
6734     \setbox\@outputbox\vbox{%
6735       \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
6736   \long\def\put(#1,#2)#3{%
6737     \@killglue
6738     % Try:
6739     \ifx\bbl@pictresetdir\relax
6740       \def\bbl@tempc{0}%
6741     \else
6742       \directlua{
6743         Babel.get_picture_dir = true
6744         Babel.picture_has_bidi = 0
6745       }%
6746       \setbox\z@\hb@xt@\z@{%
6747         \@defaultunitsset\@tempdimc{#1}\unitlength
6748         \kern\@tempdimc
6749         #3\hss}% TODO: #3 executed twice (below). That's bad.
6750       \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6751     \fi
6752     % Do:
6753     \@defaultunitsset\@tempdimc{#2}\unitlength
6754     \raise\@tempdimc\hb@xt@\z@{%
6755       \@defaultunitsset\@tempdimc{#1}\unitlength
6756       \kern\@tempdimc
6757       {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
6758     \ignorespaces}%
6759   \MakeRobust\put}%
6760 \AtBeginDocument
6761 {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir@gobble}%
6762 \ifx\pgfpicture\undefined\else % TODO. Allow deactivate?
6763   \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6764   \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6765   \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6766 \fi
6767 \ifx\tikzpicture\undefined\else
6768   \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\tw}%
6769   \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6770   \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw}%
6771 \fi
6772 \ifx\tcolorbox\undefined\else
6773   \def\tcb@drawing@env@begin{%
6774     \csname tcb@before@tcb@split@state\endcsname
6775     \bbl@pictsetdir\tw@
6776     \begin{\kvtcb@graphenv}%
6777     \tcb@bbdraw%
6778     \tcb@apply@graph@patches
6779   }%

```

```

6780 \def\tcb@drawing@env@end{%
6781 \end{\kv tcb@graphenv}%
6782 \bbl@pictresetdir
6783 \csname tcb@after@\tcb@split@state\endcsname
6784 }%
6785 \fi
6786 }}
6787 {}

```

```

6788 \IfBabelLayout{counters*}%
6789   {\bbl@add\bbl@opt@layout{.counters.}%
6790     \directlua{
6791       luatexbase.add_to_callback("process_output_buffer",
6792         Babel.discard_subl_r , "Babel.discard_subl_r") }%
6793   }}
6794 \IfBabelLayout{counters}%
6795   {\let\bbl@0L@@@textsuperscript@@textsuperscript
6796     \bbl@sreplace\textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6797     \let\bbl@latinarabic=@arabic
6798     \let\bbl@0L@@arabic@arabic
6799     \def@@arabic#1{\babelsubl_r{\bbl@latinarabic#1}}%
6800     \@ifpackagewith{babel}{\bidi=default}%
6801     {\let\bbl@asciroman=@roman
6802       \let\bbl@0L@@roman@roman
6803       \def@@roman#1{\babelsubl_r{\ensureascii{\bbl@asciroman#1}}}%
6804       \let\bbl@asciiRoman=@Roman
6805       \let\bbl@0L@@Roman@Roman
6806       \def@@Roman#1{\babelsubl_r{\ensureascii{\bbl@asciiRoman#1}}}%
6807       \let\bbl@0L@labelenumii\labelenumii
6808       \def\labelenumii{}\theenumii{}%
6809       \let\bbl@0L@p@enumiii@p@enumiii
6810       \def\p@enumiii{\p@enumii}\theenumii{}}{}{}{}
6811 \langle\langle Footnote changes \rangle\rangle
6812 \IfBabelLayout{footnotes}%
6813   {\let\bbl@0L@footnote\footnote
6814     \BabelFootnote\footnote\languagename{}}{}%
6815     \BabelFootnote\localfootnote\languagename{}}{}%
6816     \BabelFootnote\mainfootnote{}}{}{}{}
6817   {}

```

```

6818 \IfBabelLayout{extras}%
6819   {\bbl@ncarg\let\bbl@OL@underline{underline }%
6820     \bbl@carg\bbl@sreplace{underline }%
6821       {\$@@underline}{\bgroup\bbl@nextfake$@@underline}%
6822     \bbl@carg\bbl@sreplace{underline }%
6823       {\m@th$}{\m@th$\egroup}%
6824     \let\bbl@OL@LaTeXe\LaTeXe
6825     \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6826       \if b\expandafter\car\@f@series\@nil\boldmath\fi
6827       \babelsublr{%
6828         \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}%
6829   }
6830 \end{luatex}

```

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.

```

6831 (*transforms)
6832 Babel.linebreaking.replacements = {}
6833 Babel.linebreaking.replacements[0] = {} -- pre
6834 Babel.linebreaking.replacements[1] = {} -- post
6835
6836 -- Discretionaries contain strings as nodes
6837 function Babel.str_to_nodes(fn, matches, base)
6838   local n, head, last
6839   if fn == nil then return nil end
6840   for s in string.utfvalues(fn(matches)) do
6841     if base.id == 7 then
6842       base = base.replace
6843     end
6844     n = node.copy(base)
6845     n.char = s
6846     if not head then
6847       head = n
6848     else
6849       last.next = n
6850     end
6851     last = n
6852   end
6853   return head
6854 end
6855
6856 Babel.fetch_subtext = {}
6857
6858 Babel.ignore_pre_char = function(node)
6859   return (node.lang == Babel.nohyphenation)
6860 end
6861
6862 -- Merging both functions doesn't seem feasible, because there are too
6863 -- many differences.
6864 Babel.fetch_subtext[0] = function(head)
6865   local word_string = ''
6866   local word_nodes = {}
6867   local lang
6868   local item = head
6869   local inmath = false
6870
6871   while item do
6872
6873     if item.id == 11 then
6874       inmath = (item.subtype == 0)
6875     end
6876
6877     if inmath then
6878       -- pass
6879
6880     elseif item.id == 29 then
6881       local locale = node.get_attribute(item, Babel.attr_locale)
6882
6883       if lang == locale or lang == nil then
6884         lang = lang or locale

```

```

6885         if Babel.ignore_pre_char(item) then
6886             word_string = word_string .. Babel.us_char
6887         else
6888             word_string = word_string .. unicode.utf8.char(item.char)
6889         end
6890         word_nodes[#word_nodes+1] = item
6891     else
6892         break
6893     end
6894
6895     elseif item.id == 12 and item.subtype == 13 then
6896         word_string = word_string .. ' '
6897         word_nodes[#word_nodes+1] = item
6898
6899     -- Ignore leading unrecognized nodes, too.
6900     elseif word_string ~= '' then
6901         word_string = word_string .. Babel.us_char
6902         word_nodes[#word_nodes+1] = item -- Will be ignored
6903     end
6904
6905     item = item.next
6906 end
6907
6908 -- Here and above we remove some trailing chars but not the
6909 -- corresponding nodes. But they aren't accessed.
6910 if word_string:sub(-1) == ' ' then
6911     word_string = word_string:sub(1,-2)
6912 end
6913 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6914 return word_string, word_nodes, item, lang
6915 end
6916
6917 Babel.fetch_subtext[1] = function(head)
6918     local word_string = ''
6919     local word_nodes = {}
6920     local lang
6921     local item = head
6922     local inmath = false
6923
6924     while item do
6925
6926         if item.id == 11 then
6927             inmath = (item.subtype == 0)
6928         end
6929
6930         if inmath then
6931             -- pass
6932
6933         elseif item.id == 29 then
6934             if item.lang == lang or lang == nil then
6935                 if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6936                     lang = lang or item.lang
6937                     word_string = word_string .. unicode.utf8.char(item.char)
6938                     word_nodes[#word_nodes+1] = item
6939                 end
6940             else
6941                 break
6942             end
6943
6944         elseif item.id == 7 and item.subtype == 2 then
6945             word_string = word_string .. '='
6946             word_nodes[#word_nodes+1] = item
6947

```

```

6948     elseif item.id == 7 and item.subtype == 3 then
6949         word_string = word_string .. '|'
6950         word_nodes[#word_nodes+1] = item
6951
6952     -- (1) Go to next word if nothing was found, and (2) implicitly
6953     -- remove leading USs.
6954     elseif word_string == '' then
6955         -- pass
6956
6957     -- This is the responsible for splitting by words.
6958     elseif (item.id == 12 and item.subtype == 13) then
6959         break
6960
6961     else
6962         word_string = word_string .. Babel.us_char
6963         word_nodes[#word_nodes+1] = item -- Will be ignored
6964     end
6965
6966     item = item.next
6967 end
6968
6969 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6970 return word_string, word_nodes, item, lang
6971 end
6972
6973 function Babel.pre_hyphenate_replace(head)
6974     Babel.hyphenate_replace(head, 0)
6975 end
6976
6977 function Babel.post_hyphenate_replace(head)
6978     Babel.hyphenate_replace(head, 1)
6979 end
6980
6981 Babel.us_char = string.char(31)
6982
6983 function Babel.hyphenate_replace(head, mode)
6984     local u = unicode.utf8
6985     local lbkr = Babel.linebreaking.replacements[mode]
6986
6987     local word_head = head
6988
6989     while true do -- for each subtext block
6990
6991         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6992
6993         if Babel.debug then
6994             print()
6995             print((mode == 0) and '@@@<' or '@@@>', w)
6996         end
6997
6998         if nw == nil and w == '' then break end
6999
7000         if not lang then goto next end
7001         if not lbkr[lang] then goto next end
7002
7003         -- For each saved (pre|post)hyphenation. TODO. Reconsider how
7004         -- loops are nested.
7005         for k=1, #lbkr[lang] do
7006             local p = lbkr[lang][k].pattern
7007             local r = lbkr[lang][k].replace
7008             local attr = lbkr[lang][k].attr or -1
7009
7010             if Babel.debug then

```

```

7011     print('*****', p, mode)
7012 end
7013
7014 -- This variable is set in some cases below to the first *byte*
7015 -- after the match, either as found by u.match (faster) or the
7016 -- computed position based on sc if w has changed.
7017 local last_match = 0
7018 local step = 0
7019
7020 -- For every match.
7021 while true do
7022     if Babel.debug then
7023         print('====')
7024     end
7025     local new -- used when inserting and removing nodes
7026
7027     local matches = { u.match(w, p, last_match) }
7028
7029     if #matches < 2 then break end
7030
7031     -- Get and remove empty captures (with ())'s, which return a
7032     -- number with the position), and keep actual captures
7033     -- (from (...)), if any, in matches.
7034     local first = table.remove(matches, 1)
7035     local last = table.remove(matches, #matches)
7036     -- Non re-fetched substrings may contain \31, which separates
7037     -- subsubstrings.
7038     if string.find(w:sub(first, last-1), Babel.us_char) then break end
7039
7040     local save_last = last -- with A()BC()D, points to D
7041
7042     -- Fix offsets, from bytes to unicode. Explained above.
7043     first = u.len(w:sub(1, first-1)) + 1
7044     last = u.len(w:sub(1, last-1)) -- now last points to C
7045
7046     -- This loop stores in a small table the nodes
7047     -- corresponding to the pattern. Used by 'data' to provide a
7048     -- predictable behavior with 'insert' (w_nodes is modified on
7049     -- the fly), and also access to 'remove'd nodes.
7050     local sc = first-1 -- Used below, too
7051     local data_nodes = {}
7052
7053     local enabled = true
7054     for q = 1, last-first+1 do
7055         data_nodes[q] = w_nodes[sc+q]
7056         if enabled
7057             and attr > -1
7058             and not node.has_attribute(data_nodes[q], attr)
7059         then
7060             enabled = false
7061         end
7062     end
7063
7064     -- This loop traverses the matched substring and takes the
7065     -- corresponding action stored in the replacement list.
7066     -- sc = the position in substr nodes / string
7067     -- rc = the replacement table index
7068     local rc = 0
7069
7070     while rc < last-first+1 do -- for each replacement
7071         if Babel.debug then
7072             print('.....', rc + 1)
7073         end

```

```

7074         sc = sc + 1
7075         rc = rc + 1
7076
7077     if Babel.debug then
7078         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7079         local ss = ''
7080         for itt in node.traverse(head) do
7081             if itt.id == 29 then
7082                 ss = ss .. unicode.utf8.char(itt.char)
7083             else
7084                 ss = ss .. '{' .. itt.id .. '}'
7085             end
7086         end
7087         print('*****', ss)
7088
7089     end
7090
7091     local crep = r[rc]
7092     local item = w_nodes[sc]
7093     local item_base = item
7094     local placeholder = Babel.us_char
7095     local d
7096
7097     if crep and crep.data then
7098         item_base = data_nodes[crep.data]
7099     end
7100
7101     if crep then
7102         step = crep.step or 0
7103     end
7104
7105     if (not enabled) or (crep and next(crep) == nil) then -- = {}
7106         last_match = save_last -- Optimization
7107         goto next
7108
7109     elseif crep == nil or crep.remove then
7110         node.remove(head, item)
7111         table.remove(w_nodes, sc)
7112         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7113         sc = sc - 1 -- Nothing has been inserted.
7114         last_match = utf8.offset(w, sc+1+step)
7115         goto next
7116
7117     elseif crep and crep.kashida then -- Experimental
7118         node.set_attribute(item,
7119             Babel.attr_kashida,
7120             crep.kashida)
7121         last_match = utf8.offset(w, sc+1+step)
7122         goto next
7123
7124     elseif crep and crep.string then
7125         local str = crep.string(matches)
7126         if str == '' then -- Gather with nil
7127             node.remove(head, item)
7128             table.remove(w_nodes, sc)
7129             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7130             sc = sc - 1 -- Nothing has been inserted.
7131         else
7132             local loop_first = true
7133             for s in string.utfvalues(str) do
7134                 d = node.copy(item_base)
7135                 d.char = s
7136                 if loop_first then

```

```

7137         loop_first = false
7138         head, new = node.insert_before(head, item, d)
7139         if sc == 1 then
7140             word_head = head
7141         end
7142         w_nodes[sc] = d
7143         w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
7144     else
7145         sc = sc + 1
7146         head, new = node.insert_before(head, item, d)
7147         table.insert(w_nodes, sc, new)
7148         w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7149     end
7150     if Babel.debug then
7151         print('.....', 'str')
7152         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7153     end
7154     end -- for
7155     node.remove(head, item)
7156 end -- if ''
7157 last_match = utf8.offset(w, sc+1+step)
7158 goto next
7159
7160 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7161     d = node.new(7, 3) -- (disc, regular)
7162     d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7163     d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7164     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7165     d.attr = item_base.attr
7166     if crep.pre == nil then -- TeXbook p96
7167         d.penalty = crep.penalty or tex.hyphenpenalty
7168     else
7169         d.penalty = crep.penalty or tex.exhyphenpenalty
7170     end
7171     placeholder = '|'
7172     head, new = node.insert_before(head, item, d)
7173
7174 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7175     -- ERROR
7176
7177 elseif crep and crep.penalty then
7178     d = node.new(14, 0) -- (penalty, userpenalty)
7179     d.attr = item_base.attr
7180     d.penalty = crep.penalty
7181     head, new = node.insert_before(head, item, d)
7182
7183 elseif crep and crep.space then
7184     -- 655360 = 10 pt = 10 * 65536 sp
7185     d = node.new(12, 13) -- (glue, spaceskip)
7186     local quad = font.getfont(item_base.font).size or 655360
7187     node.setglue(d, crep.space[1] * quad,
7188                 crep.space[2] * quad,
7189                 crep.space[3] * quad)
7190     if mode == 0 then
7191         placeholder = ' '
7192     end
7193     head, new = node.insert_before(head, item, d)
7194
7195 elseif crep and crep.spacefactor then
7196     d = node.new(12, 13) -- (glue, spaceskip)
7197     local base_font = font.getfont(item_base.font)
7198     node.setglue(d,
7199                 crep.spacefactor[1] * base_font.parameters['space'],

```



```

7200         crep.spacefactor[2] * base_font.parameters['space_stretch'],
7201         crep.spacefactor[3] * base_font.parameters['space_shrink'])
7202     if mode == 0 then
7203         placeholder = ' '
7204     end
7205     head, new = node.insert_before(head, item, d)
7206
7207     elseif mode == 0 and crep and crep.space then
7208         -- ERROR
7209
7210     end -- ie replacement cases
7211
7212     -- Shared by disc, space and penalty.
7213     if sc == 1 then
7214         word_head = head
7215     end
7216     if crep.insert then
7217         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7218         table.insert(w_nodes, sc, new)
7219         last = last + 1
7220     else
7221         w_nodes[sc] = d
7222         node.remove(head, item)
7223         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7224     end
7225
7226     last_match = utf8.offset(w, sc+1+step)
7227
7228     ::next::
7229
7230 end -- for each replacement
7231
7232 if Babel.debug then
7233     print('.....', '/')
7234     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7235 end
7236
7237 end -- for match
7238
7239 end -- for patterns
7240
7241 ::next::
7242 word_head = nw
7243 end -- for substring
7244 return head
7245 end
7246
7247 -- This table stores capture maps, numbered consecutively
7248 Babel.capture_maps = {}
7249
7250 -- The following functions belong to the next macro
7251 function Babel.capture_func(key, cap)
7252     local ret = "[" .. cap:gsub('{{[0-9]}}', ")]..m[%1]..["] .. "]"
7253     local cnt
7254     local u = unicode.utf8
7255     ret, cnt = ret:gsub('{{[0-9]}|([^\]]+)|(.-)}', Babel.capture_func_map)
7256     if cnt == 0 then
7257         ret = u.gsub(ret, '{(%x%x%x%x+)}',
7258             function (n)
7259                 return u.char(tonumber(n, 16))
7260             end)
7261     end
7262     ret = ret:gsub("%[%[%]%]%.%", '')

```

```

7263 ret = ret:gsub("%.%.%[%[%]]", '')
7264 return key .. [[=function(m) return ]] .. ret .. [[ end]]
7265 end
7266
7267 function Babel.capt_map(from, mapno)
7268 return Babel.capture_maps[mapno][from] or from
7269 end
7270
7271 -- Handle the {n|abc|ABC} syntax in captures
7272 function Babel.capture_func_map(capno, from, to)
7273 local u = unicode.utf8
7274 from = u.gsub(from, '{(%x%x%x%x+)}',
7275 function (n)
7276 return u.char(tonumber(n, 16))
7277 end)
7278 to = u.gsub(to, '{(%x%x%x%x+)}',
7279 function (n)
7280 return u.char(tonumber(n, 16))
7281 end)
7282 local froms = {}
7283 for s in string.utfcharacters(from) do
7284 table.insert(froms, s)
7285 end
7286 local cnt = 1
7287 table.insert(Babel.capture_maps, {})
7288 local mlen = table.getn(Babel.capture_maps)
7289 for s in string.utfcharacters(to) do
7290 Babel.capture_maps[mlen][froms[cnt]] = s
7291 cnt = cnt + 1
7292 end
7293 return "]]..Babel.capt_map(m[" .. capno .. "], " ..
7294 (mlen) .. ").." .. "["
7295 end
7296
7297 -- Create/Extend reversed sorted list of kashida weights:
7298 function Babel.capture_kashida(key, wt)
7299 wt = tonumber(wt)
7300 if Babel.kashida_wts then
7301 for p, q in ipairs(Babel.kashida_wts) do
7302 if wt == q then
7303 break
7304 elseif wt > q then
7305 table.insert(Babel.kashida_wts, p, wt)
7306 break
7307 elseif table.getn(Babel.kashida_wts) == p then
7308 table.insert(Babel.kashida_wts, wt)
7309 end
7310 end
7311 else
7312 Babel.kashida_wts = { wt }
7313 end
7314 return 'kashida = ' .. wt
7315 end
7316
7317 -- Experimental: applies prehyphenation transforms to a string (letters
7318 -- and spaces).
7319 function Babel.string_prehyphenation(str, locale)
7320 local n, head, last, res
7321 head = node.new(8, 0) -- dummy (hack just to start)
7322 last = head
7323 for s in string.utfvalues(str) do
7324 if s == 20 then
7325 n = node.new(12, 0)

```

```

7326     else
7327         n = node.new(29, 0)
7328         n.char = s
7329     end
7330     node.set_attribute(n, Babel.attr_locale, locale)
7331     last.next = n
7332     last = n
7333 end
7334 head = Babel.hyphenate_replace(head, 0)
7335 res = ''
7336 for n in node.traverse(head) do
7337     if n.id == 12 then
7338         res = res .. ' '
7339     elseif n.id == 29 then
7340         res = res .. unicode.utf8.char(n.char)
7341     end
7342 end
7343 tex.print(res)
7344 end
7345 </transforms>

```

10.12 Lua: Auto bidi with basic and basic-r

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

```

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

```

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

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

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

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

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

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

```

7346 (*basic-r)
7347 Babel = Babel or {}
7348
7349 Babel.bidi_enabled = true

```

```

7350
7351 require('babel-data-bidi.lua')
7352
7353 local characters = Babel.characters
7354 local ranges = Babel.ranges
7355
7356 local DIR = node.id("dir")
7357
7358 local function dir_mark(head, from, to, outer)
7359   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
7360   local d = node.new(DIR)
7361   d.dir = '+' .. dir
7362   node.insert_before(head, from, d)
7363   d = node.new(DIR)
7364   d.dir = '-' .. dir
7365   node.insert_after(head, to, d)
7366 end
7367
7368 function Babel.bidi(head, ispar)
7369   local first_n, last_n          -- first and last char with nums
7370   local last_es                  -- an auxiliary 'last' used with nums
7371   local first_d, last_d          -- first and last char in L/R block
7372   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):

```

7373   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7374   local strong_lr = (strong == 'l') and 'l' or 'r'
7375   local outer = strong
7376
7377   local new_dir = false
7378   local first_dir = false
7379   local inmath = false
7380
7381   local last_lr
7382
7383   local type_n = ''
7384
7385   for item in node.traverse(head) do
7386
7387     -- three cases: glyph, dir, otherwise
7388     if item.id == node.id'glyph'
7389       or (item.id == 7 and item.subtype == 2) then
7390
7391       local itemchar
7392       if item.id == 7 and item.subtype == 2 then
7393         itemchar = item.replace.char
7394       else
7395         itemchar = item.char
7396       end
7397       local chardata = characters[itemchar]
7398       dir = chardata and chardata.d or nil
7399       if not dir then
7400         for nn, et in ipairs(ranges) do
7401           if itemchar < et[1] then
7402             break
7403           elseif itemchar <= et[2] then
7404             dir = et[3]
7405             break
7406           end
7407         end
7408       end

```

```

7409     dir = dir or 'l'
7410     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).

```

7411     if new_dir then
7412         attr_dir = 0
7413         for at in node.traverse(item.attr) do
7414             if at.number == Babel.attr_dir then
7415                 attr_dir = at.value & 0x3
7416             end
7417         end
7418         if attr_dir == 1 then
7419             strong = 'r'
7420         elseif attr_dir == 2 then
7421             strong = 'al'
7422         else
7423             strong = 'l'
7424         end
7425         strong_lr = (strong == 'l') and 'l' or 'r'
7426         outer = strong_lr
7427         new_dir = false
7428     end
7429
7430     if dir == 'nsm' then dir = strong end -- W1

```

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

```

7431     dir_real = dir -- We need dir_real to set strong below
7432     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:

```

7433     if strong == 'al' then
7434         if dir == 'en' then dir = 'an' end -- W2
7435         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7436         strong_lr = 'r' -- W3
7437     end

```

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

```

7438     elseif item.id == node.id'dir' and not inmath then
7439         new_dir = true
7440         dir = nil
7441     elseif item.id == node.id'math' then
7442         inmath = (item.subtype == 0)
7443     else
7444         dir = nil -- Not a char
7445     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>.

```

7446     if dir == 'en' or dir == 'an' or dir == 'et' then
7447         if dir ~= 'et' then
7448             type_n = dir
7449         end
7450         first_n = first_n or item
7451         last_n = last_es or item
7452         last_es = nil
7453     elseif dir == 'es' and last_n then -- W3+W6

```

```

7454     last_es = item
7455     elseif dir == 'cs' then          -- it's right - do nothing
7456     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7457         if strong_lr == 'r' and type_n ~= '' then
7458             dir_mark(head, first_n, last_n, 'r')
7459         elseif strong_lr == 'l' and first_d and type_n == 'an' then
7460             dir_mark(head, first_n, last_n, 'r')
7461             dir_mark(head, first_d, last_d, outer)
7462             first_d, last_d = nil, nil
7463         elseif strong_lr == 'l' and type_n ~= '' then
7464             last_d = last_n
7465         end
7466         type_n = ''
7467         first_n, last_n = nil, nil
7468     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:

```

7469     if dir == 'l' or dir == 'r' then
7470         if dir ~= outer then
7471             first_d = first_d or item
7472             last_d = item
7473         elseif first_d and dir ~= strong_lr then
7474             dir_mark(head, first_d, last_d, outer)
7475             first_d, last_d = nil, nil
7476         end
7477     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.

```

7478     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7479         item.char = characters[item.char] and
7480             characters[item.char].m or item.char
7481     elseif (dir or new_dir) and last_lr ~= item then
7482         local mir = outer .. strong_lr .. (dir or outer)
7483         if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7484             for ch in node.traverse(node.next(last_lr)) do
7485                 if ch == item then break end
7486                 if ch.id == 'glyph' and characters[ch.char] then
7487                     ch.char = characters[ch.char].m or ch.char
7488                 end
7489             end
7490         end
7491     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).

```

7492     if dir == 'l' or dir == 'r' then
7493         last_lr = item
7494         strong = dir_real          -- Don't search back - best save now
7495         strong_lr = (strong == 'l') and 'l' or 'r'
7496     elseif new_dir then
7497         last_lr = nil
7498     end
7499 end

```

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

```

7500     if last_lr and outer == 'r' then
7501         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do

```

```

7502     if characters[ch.char] then
7503         ch.char = characters[ch.char].m or ch.char
7504     end
7505 end
7506 end
7507 if first_n then
7508     dir_mark(head, first_n, last_n, outer)
7509 end
7510 if first_d then
7511     dir_mark(head, first_d, last_d, outer)
7512 end

```

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

```

7513 return node.prev(head) or head
7514 end
7515 </basic-r>

```

And here the Lua code for bidi=basic:

```

7516 <(*basic)
7517 Babel = Babel or {}
7518
7519 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
7520
7521 Babel.fontmap = Babel.fontmap or {}
7522 Babel.fontmap[0] = {}      -- l
7523 Babel.fontmap[1] = {}      -- r
7524 Babel.fontmap[2] = {}      -- al/an
7525
7526 Babel.bidi_enabled = true
7527 Babel.mirroring_enabled = true
7528
7529 require('babel-data-bidi.lua')
7530
7531 local characters = Babel.characters
7532 local ranges = Babel.ranges
7533
7534 local DIR = node.id('dir')
7535 local GLYPH = node.id('glyph')
7536
7537 local function insert_implicit(head, state, outer)
7538     local new_state = state
7539     if state.sim and state.eim and state.sim ~= state.eim then
7540         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
7541         local d = node.new(DIR)
7542         d.dir = '+' .. dir
7543         node.insert_before(head, state.sim, d)
7544         local d = node.new(DIR)
7545         d.dir = '-' .. dir
7546         node.insert_after(head, state.eim, d)
7547     end
7548     new_state.sim, new_state.eim = nil, nil
7549     return head, new_state
7550 end
7551
7552 local function insert_numeric(head, state)
7553     local new
7554     local new_state = state
7555     if state.san and state.ean and state.san ~= state.ean then
7556         local d = node.new(DIR)
7557         d.dir = '+TLT'
7558         _, new = node.insert_before(head, state.san, d)
7559         if state.san == state.sim then state.sim = new end
7560         local d = node.new(DIR)

```

```

7561     d.dir = '-TLT'
7562     _, new = node.insert_after(head, state.ean, d)
7563     if state.ean == state.eim then state.eim = new end
7564 end
7565 new_state.san, new_state.ean = nil, nil
7566 return head, new_state
7567 end
7568
7569 -- TODO - \hbox with an explicit dir can lead to wrong results
7570 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7571 -- was s made to improve the situation, but the problem is the 3-dir
7572 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7573 -- well.
7574
7575 function Babel.bidi(head, ispar, hdir)
7576     local d -- d is used mainly for computations in a loop
7577     local prev_d = ''
7578     local new_d = false
7579
7580     local nodes = {}
7581     local outer_first = nil
7582     local inmath = false
7583
7584     local glue_d = nil
7585     local glue_i = nil
7586
7587     local has_en = false
7588     local first_et = nil
7589
7590     local has_hyperlink = false
7591
7592     local ATDIR = Babel.attr_dir
7593
7594     local save_outer
7595     local temp = node.get_attribute(head, ATDIR)
7596     if temp then
7597         temp = temp & 0x3
7598         save_outer = (temp == 0 and 'l') or
7599                     (temp == 1 and 'r') or
7600                     (temp == 2 and 'al')
7601     elseif ispar then -- Or error? Shouldn't happen
7602         save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7603     else -- Or error? Shouldn't happen
7604         save_outer = ('TRT' == hdir) and 'r' or 'l'
7605     end
7606     -- when the callback is called, we are just _after_ the box,
7607     -- and the textdir is that of the surrounding text
7608     -- if not ispar and hdir ~= tex.textdir then
7609     --     save_outer = ('TRT' == hdir) and 'r' or 'l'
7610     -- end
7611     local outer = save_outer
7612     local last = outer
7613     -- 'al' is only taken into account in the first, current loop
7614     if save_outer == 'al' then save_outer = 'r' end
7615
7616     local fontmap = Babel.fontmap
7617
7618     for item in node.traverse(head) do
7619
7620         -- In what follows, #node is the last (previous) node, because the
7621         -- current one is not added until we start processing the neutrals.
7622
7623         -- three cases: glyph, dir, otherwise

```



```

7624 if item.id == GLYPH
7625     or (item.id == 7 and item.subtype == 2) then
7626
7627     local d_font = nil
7628     local item_r
7629     if item.id == 7 and item.subtype == 2 then
7630         item_r = item.replace    -- automatic discs have just 1 glyph
7631     else
7632         item_r = item
7633     end
7634     local chardata = characters[item_r.char]
7635     d = chardata and chardata.d or nil
7636     if not d or d == 'nsm' then
7637         for nn, et in ipairs(ranges) do
7638             if item_r.char < et[1] then
7639                 break
7640             elseif item_r.char <= et[2] then
7641                 if not d then d = et[3]
7642                 elseif d == 'nsm' then d_font = et[3]
7643                 end
7644                 break
7645             end
7646         end
7647     end
7648     d = d or 'l'
7649
7650     -- A short 'pause' in bidi for mapfont
7651     d_font = d_font or d
7652     d_font = (d_font == 'l' and 0) or
7653             (d_font == 'nsm' and 0) or
7654             (d_font == 'r' and 1) or
7655             (d_font == 'al' and 2) or
7656             (d_font == 'an' and 2) or nil
7657     if d_font and fontmap and fontmap[d_font][item_r.font] then
7658         item_r.font = fontmap[d_font][item_r.font]
7659     end
7660
7661     if new_d then
7662         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7663         if inmath then
7664             attr_d = 0
7665         else
7666             attr_d = node.get_attribute(item, ATDIR)
7667             attr_d = attr_d & 0x3
7668         end
7669         if attr_d == 1 then
7670             outer_first = 'r'
7671             last = 'r'
7672         elseif attr_d == 2 then
7673             outer_first = 'r'
7674             last = 'al'
7675         else
7676             outer_first = 'l'
7677             last = 'l'
7678         end
7679         outer = last
7680         has_en = false
7681         first_et = nil
7682         new_d = false
7683     end
7684
7685     if glue_d then
7686         if (d == 'l' and 'l' or 'r') ~= glue_d then

```

```

7687         table.insert(nodes, {glue_i, 'on', nil})
7688     end
7689     glue_d = nil
7690     glue_i = nil
7691 end
7692
7693 elseif item.id == DIR then
7694     d = nil
7695
7696     if head ~= item then new_d = true end
7697
7698     elseif item.id == node.id'glue' and item.subtype == 13 then
7699         glue_d = d
7700         glue_i = item
7701         d = nil
7702
7703     elseif item.id == node.id'math' then
7704         inmath = (item.subtype == 0)
7705
7706     elseif item.id == 8 and item.subtype == 19 then
7707         has_hyperlink = true
7708
7709     else
7710         d = nil
7711     end
7712
7713     -- AL <= EN/ET/ES      -- W2 + W3 + W6
7714     if last == 'al' and d == 'en' then
7715         d = 'an'          -- W3
7716     elseif last == 'al' and (d == 'et' or d == 'es') then
7717         d = 'on'          -- W6
7718     end
7719
7720     -- EN + CS/ES + EN      -- W4
7721     if d == 'en' and #nodes >= 2 then
7722         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7723             and nodes[#nodes-1][2] == 'en' then
7724             nodes[#nodes][2] = 'en'
7725         end
7726     end
7727
7728     -- AN + CS + AN        -- W4 too, because uax9 mixes both cases
7729     if d == 'an' and #nodes >= 2 then
7730         if (nodes[#nodes][2] == 'cs')
7731             and nodes[#nodes-1][2] == 'an' then
7732             nodes[#nodes][2] = 'an'
7733         end
7734     end
7735
7736     -- ET/EN              -- W5 + W7->l / W6->on
7737     if d == 'et' then
7738         first_et = first_et or (#nodes + 1)
7739     elseif d == 'en' then
7740         has_en = true
7741         first_et = first_et or (#nodes + 1)
7742     elseif first_et then    -- d may be nil here !
7743         if has_en then
7744             if last == 'l' then
7745                 temp = 'l'    -- W7
7746             else
7747                 temp = 'en'   -- W5
7748             end
7749         else

```

```

7750         temp = 'on'      -- W6
7751     end
7752     for e = first_et, #nodes do
7753         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7754     end
7755     first_et = nil
7756     has_en = false
7757 end
7758
7759 -- Force mathdir in math if ON (currently works as expected only
7760 -- with 'l')
7761 if inmath and d == 'on' then
7762     d = ('TRT' == tex.mathdir) and 'r' or 'l'
7763 end
7764
7765 if d then
7766     if d == 'al' then
7767         d = 'r'
7768         last = 'al'
7769     elseif d == 'l' or d == 'r' then
7770         last = d
7771     end
7772     prev_d = d
7773     table.insert(nodes, {item, d, outer_first})
7774 end
7775
7776 outer_first = nil
7777
7778 end
7779
7780 -- TODO -- repeated here in case EN/ET is the last node. Find a
7781 -- better way of doing things:
7782 if first_et then      -- dir may be nil here !
7783     if has_en then
7784         if last == 'l' then
7785             temp = 'l'      -- W7
7786         else
7787             temp = 'en'     -- W5
7788         end
7789     else
7790         temp = 'on'        -- W6
7791     end
7792     for e = first_et, #nodes do
7793         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7794     end
7795 end
7796
7797 -- dummy node, to close things
7798 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7799
7800 ----- NEUTRAL -----
7801
7802 outer = save_outer
7803 last = outer
7804
7805 local first_on = nil
7806
7807 for q = 1, #nodes do
7808     local item
7809
7810     local outer_first = nodes[q][3]
7811     outer = outer_first or outer
7812     last = outer_first or last

```

```

7813
7814     local d = nodes[q][2]
7815     if d == 'an' or d == 'en' then d = 'r' end
7816     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7817
7818     if d == 'on' then
7819         first_on = first_on or q
7820     elseif first_on then
7821         if last == d then
7822             temp = d
7823         else
7824             temp = outer
7825         end
7826         for r = first_on, q - 1 do
7827             nodes[r][2] = temp
7828             item = nodes[r][1] -- MIRRORING
7829             if Babel.mirroring_enabled and item.id == GLYPH
7830                 and temp == 'r' and characters[item.char] then
7831                 local font_mode = ''
7832                 if item.font > 0 and font.fonts[item.font].properties then
7833                     font_mode = font.fonts[item.font].properties.mode
7834                 end
7835                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
7836                     item.char = characters[item.char].m or item.char
7837                 end
7838             end
7839         end
7840         first_on = nil
7841     end
7842
7843     if d == 'r' or d == 'l' then last = d end
7844 end
7845
7846 ----- IMPLICIT, REORDER -----
7847
7848 outer = save_outer
7849 last = outer
7850
7851 local state = {}
7852 state.has_r = false
7853
7854 for q = 1, #nodes do
7855     local item = nodes[q][1]
7856
7857     outer = nodes[q][3] or outer
7858
7859     local d = nodes[q][2]
7860
7861     if d == 'nsm' then d = last end -- W1
7862     if d == 'en' then d = 'an' end
7863     local isdir = (d == 'r' or d == 'l')
7864
7865     if outer == 'l' and d == 'an' then
7866         state.san = state.san or item
7867         state.ean = item
7868     elseif state.san then
7869         head, state = insert_numeric(head, state)
7870     end
7871
7872     if outer == 'l' then
7873         if d == 'an' or d == 'r' then -- im -> implicit
7874             if d == 'r' then state.has_r = true end

```

```

7876     state.sim = state.sim or item
7877     state.eim = item
7878     elseif d == 'l' and state.sim and state.has_r then
7879         head, state = insert_implicit(head, state, outer)
7880     elseif d == 'l' then
7881         state.sim, state.eim, state.has_r = nil, nil, false
7882     end
7883 else
7884     if d == 'an' or d == 'l' then
7885         if nodes[q][3] then -- nil except after an explicit dir
7886             state.sim = item -- so we move sim 'inside' the group
7887         else
7888             state.sim = state.sim or item
7889         end
7890         state.eim = item
7891     elseif d == 'r' and state.sim then
7892         head, state = insert_implicit(head, state, outer)
7893     elseif d == 'r' then
7894         state.sim, state.eim = nil, nil
7895     end
7896 end
7897
7898 if isdir then
7899     last = d -- Don't search back - best save now
7900 elseif d == 'on' and state.san then
7901     state.san = state.san or item
7902     state.ean = item
7903 end
7904
7905 end
7906
7907 head = node.prev(head) or head
7908
7909 ----- FIX HYPERLINKS -----
7910
7911 if has_hyperlink then
7912     local flag, linking = 0, 0
7913     for item in node.traverse(head) do
7914         if item.id == DIR then
7915             if item.dir == '+TRT' or item.dir == '+TLT' then
7916                 flag = flag + 1
7917             elseif item.dir == '-TRT' or item.dir == '-TLT' then
7918                 flag = flag - 1
7919             end
7920         elseif item.id == 8 and item.subtype == 19 then
7921             linking = flag
7922         elseif item.id == 8 and item.subtype == 20 then
7923             if linking > 0 then
7924                 if item.prev.id == DIR and
7925                     (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
7926                     d = node.new(DIR)
7927                     d.dir = item.prev.dir
7928                     node.remove(head, item.prev)
7929                     node.insert_after(head, item, d)
7930                 end
7931             end
7932             linking = 0
7933         end
7934     end
7935 end
7936
7937 return head
7938 end

```

7939 \langle /basic \rangle

11 Data for CJK

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

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

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

12 The ‘nil’ language

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

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

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

```
7940  $\langle$ *nil $\rangle$ 
7941 \ProvidesLanguage{nil}[\mathrel{\langle}date\mathrel{\rangle}] v[\mathrel{\langle}version\mathrel{\rangle}] Nil language]
7942 \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.

```
7943 \ifx\l@nil\undefined
7944   \newlanguage\l@nil
7945   \@namedef{bbl@hyphendata@the\l@nil}{{}}{}% Remove warning
7946   \let\bbl@elt\relax
7947   \edef\bbl@languages{% Add it to the list of languages
7948     \bbl@languages\bbl@elt{nil}{the\l@nil}{{}}{}
7949 \fi
```

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

```
7950 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
\datenil
7951 \let\captionnil\empty
7952 \let\datenil\empty
```

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

```
7953 \def\bbl@inidata@nil{%
7954   \bbl@elt{identification}{tag.ini}{und}%
7955   \bbl@elt{identification}{load.level}{0}%
7956   \bbl@elt{identification}{charset}{utf8}%
7957   \bbl@elt{identification}{version}{1.0}%
7958   \bbl@elt{identification}{date}{2022-05-16}%
7959   \bbl@elt{identification}{name.local}{nil}%
7960   \bbl@elt{identification}{name.english}{nil}%
7961   \bbl@elt{identification}{name.babel}{nil}%
7962   \bbl@elt{identification}{tag.bcp47}{und}%
7963   \bbl@elt{identification}{language.tag.bcp47}{und}%
7964   \bbl@elt{identification}{tag.opentype}{dflt}%
7965   \bbl@elt{identification}{script.name}{Latin}%
7966   \bbl@elt{identification}{script.tag.bcp47}{Latn}%
7967   \bbl@elt{identification}{script.tag.opentype}{DFLT}%
```

```

7968 \bbl@elt{identification}{level}{1}%
7969 \bbl@elt{identification}{encodings}{}%
7970 \bbl@elt{identification}{derivate}{no}}
7971 \@namedef{bbl@tbc@nil}{und}
7972 \@namedef{bbl@lbc@nil}{und}
7973 \@namedef{bbl@casing@nil}{und} % TODO
7974 \@namedef{bbl@lotf@nil}{dflt}
7975 \@namedef{bbl@elname@nil}{nil}
7976 \@namedef{bbl@lname@nil}{nil}
7977 \@namedef{bbl@esname@nil}{Latin}
7978 \@namedef{bbl@sname@nil}{Latin}
7979 \@namedef{bbl@sbc@nil}{Latn}
7980 \@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.

```

7981 \ldf@finish{nil}
7982 \</nil>

```

13 Calendars

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

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

```

7983 <<Compute Julian day>> ≡
7984 \def\bbl@fmod#1#2{(#1-#2*floor(#1/#2))}
7985 \def\bbl@cs@gregleap#1{%
7986   (\bbl@fmod{#1}{4} == 0) &&
7987   (!((\bbl@fmod{#1}{100} == 0) && (\bbl@fmod{#1}{400} != 0)))}
7988 \def\bbl@cs@jd#1#2#3{% year, month, day
7989   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
7990     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
7991     floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) +
7992     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3 } }
7993 <</Compute Julian day>>

```

13.1 Islamic

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

```

7994 <*ca-islamic>
7995 \ExplSyntaxOn
7996 <<Compute Julian day>>
7997 % == islamic (default)
7998 % Not yet implemented
7999 \def\bbl@ca@islamic#1-#2-#3\@@#4#5#6{

```

The Civil calendar:

```

8000 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
8001   ((#3 + ceil(29.5 * (#2 - 1)) +
8002     (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
8003     1948439.5) - 1) }
8004 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
8005 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
8006 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
8007 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
8008 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
8009 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@@#5#6#7{%
8010   \edef\bbl@tempa{%
8011     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
8012   \edef#5{%
8013     \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%

```

```

8014 \edef#6{\fp_eval:n{
8015   min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
8016 \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).

```

8017 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
8018 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
8019 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
8020 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
8021 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
8022 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
8023 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
8024 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
8025 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
8026 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
8027 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
8028 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
8029 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
8030 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8031 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8032 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8033 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
8034 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
8035 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8036 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8037 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8038 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8039 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8040 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
8041 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8042 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8043 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
8044 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8045 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8046 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8047 65401,65431,65460,65490,65520}
8048 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
8049 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
8050 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
8051 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
8052   \ifnum#2>2014 \ifnum#2<2038
8053     \bbl@afterfi\expandafter@gobble
8054   \fi\fi
8055   {\bbl@error{Year~out~of~range}{The~allowed~range~is~2014-2038}}%
8056   \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
8057     \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8058   \count@ \@ne
8059   \bbl@foreach\bbl@cs@umalqura@data{%
8060     \advance\count@\@ne
8061     \ifnum##1>\bbl@tempd\else
8062       \edef\bbl@tempe{\the\count@}%
8063       \edef\bbl@tempb{##1}%
8064     \fi}%
8065   \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
8066   \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1 ) / 12) }}% annus
8067   \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
8068   \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
8069   \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
8070 \ExplSyntaxOff
8071 \bbl@add\bbl@precalendar{%

```



```

8072 \bbl@replace\bbl@ld@calendar{-civil}{}%
8073 \bbl@replace\bbl@ld@calendar{-umalqura}{}%
8074 \bbl@replace\bbl@ld@calendar{+}{}%
8075 \bbl@replace\bbl@ld@calendar{-}{%}
8076 </ca-islamic>

```

13.2 Hebrew

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

```

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

```

```

8129 #2=\tmpc
8130 \tmpc=\tmpb
8131 \divide \tmpc by 4
8132 \advance #2 by \tmpc
8133 \tmpc=\tmpb
8134 \divide \tmpc by 100
8135 \advance #2 by -\tmpc
8136 \tmpc=\tmpb
8137 \divide \tmpc by 400
8138 \advance #2 by \tmpc
8139 \global\bbl@cntcommon=#2\relax}%
8140 #2=\bbl@cntcommon}
8141 \def\bbl@absfromgreg#1#2#3#4{%
8142 {\countdef\tmpd=0
8143 #4=#1\relax
8144 \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
8145 \advance #4 by \tmpd
8146 \bbl@gregdaysprioryears{#3}{\tmpd}%
8147 \advance #4 by \tmpd
8148 \global\bbl@cntcommon=#4\relax}%
8149 #4=\bbl@cntcommon}
8150 \newif\ifbbl@hebrleap
8151 \def\bbl@checkleaphebryear#1{%
8152 {\countdef\tmpa=0
8153 \countdef\tmpb=1
8154 \tmpa=#1\relax
8155 \multiply \tmpa by 7
8156 \advance \tmpa by 1
8157 \bbl@remainder{\tmpa}{19}{\tmpb}%
8158 \ifnum \tmpb < 7
8159 \global\bbl@hebrleaptrue
8160 \else
8161 \global\bbl@hebrleapfalse
8162 \fi}}
8163 \def\bbl@hebreleapsedmonths#1#2{%
8164 {\countdef\tmpa=0
8165 \countdef\tmpb=1
8166 \countdef\tmpc=2
8167 \tmpa=#1\relax
8168 \advance \tmpa by -1
8169 #2=\tmpa
8170 \divide #2 by 19
8171 \multiply #2 by 235
8172 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8173 \tmpc=\tmpb
8174 \multiply \tmpb by 12
8175 \advance #2 by \tmpb
8176 \multiply \tmpc by 7
8177 \advance \tmpc by 1
8178 \divide \tmpc by 19
8179 \advance #2 by \tmpc
8180 \global\bbl@cntcommon=#2}%
8181 #2=\bbl@cntcommon}
8182 \def\bbl@hebreleapseddays#1#2{%
8183 {\countdef\tmpa=0
8184 \countdef\tmpb=1
8185 \countdef\tmpc=2
8186 \bbl@hebreleapsedmonths{#1}{#2}%
8187 \tmpa=#2\relax
8188 \multiply \tmpa by 13753
8189 \advance \tmpa by 5604
8190 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8191 \divide \tmpa by 25920

```

```

8192 \multiply #2 by 29
8193 \advance #2 by 1
8194 \advance #2 by \tmpa
8195 \bbl@remainder{#2}{7}{\tmpa}%
8196 \ifnum \tmpc < 19440
8197   \ifnum \tmpc < 9924
8198     \else
8199       \ifnum \tmpa=2
8200         \bbl@checkleaphebrewyear{#1}% of a common year
8201         \ifbbl@hebrleap
8202           \else
8203             \advance #2 by 1
8204           \fi
8205         \fi
8206       \fi
8207       \ifnum \tmpc < 16789
8208       \else
8209         \ifnum \tmpa=1
8210           \advance #1 by -1
8211           \bbl@checkleaphebrewyear{#1}% at the end of leap year
8212           \ifbbl@hebrleap
8213             \advance #2 by 1
8214           \fi
8215         \fi
8216       \fi
8217     \else
8218       \advance #2 by 1
8219     \fi
8220 \bbl@remainder{#2}{7}{\tmpa}%
8221 \ifnum \tmpa=0
8222   \advance #2 by 1
8223 \else
8224   \ifnum \tmpa=3
8225     \advance #2 by 1
8226   \else
8227     \ifnum \tmpa=5
8228       \advance #2 by 1
8229     \fi
8230   \fi
8231 \fi
8232 \global\bbl@cntcommon=#2\relax}%
8233 #2=\bbl@cntcommon}
8234 \def\bbl@daysinhebrewyear#1#2{%
8235   {\countdef\tmpe=12
8236   \bbl@hebreleapseddays{#1}{\tmpe}%
8237   \advance #1 by 1
8238   \bbl@hebreleapseddays{#1}{#2}%
8239   \advance #2 by -\tmpe
8240   \global\bbl@cntcommon=#2}%
8241   #2=\bbl@cntcommon}
8242 \def\bbl@hebrdayspriormonths#1#2#3{%
8243   {\countdef\tmpf= 14
8244   #3=\ifcase #1\relax
8245     0 \or
8246     0 \or
8247     30 \or
8248     59 \or
8249     89 \or
8250     118 \or
8251     148 \or
8252     148 \or
8253     177 \or
8254     207 \or

```

```

8255         236 \or
8256         266 \or
8257         295 \or
8258         325 \or
8259         400
8260     \fi
8261     \bbl@checkleaphebryear{#2}%
8262     \ifbbl@hebrleap
8263         \ifnum #1 > 6
8264             \advance #3 by 30
8265         \fi
8266     \fi
8267     \bbl@daysinhebryear{#2}{\tmpf}%
8268     \ifnum #1 > 3
8269         \ifnum \tmpf=353
8270             \advance #3 by -1
8271         \fi
8272         \ifnum \tmpf=383
8273             \advance #3 by -1
8274         \fi
8275     \fi
8276     \ifnum #1 > 2
8277         \ifnum \tmpf=355
8278             \advance #3 by 1
8279         \fi
8280         \ifnum \tmpf=385
8281             \advance #3 by 1
8282         \fi
8283     \fi
8284     \global\bbl@cntcommon=#3\relax}%
8285     #3=\bbl@cntcommon}
8286 \def\bbl@absfromhebr#1#2#3#4{%
8287     {#4=#1\relax
8288     \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8289     \advance #4 by #1\relax
8290     \bbl@hebrrelapseddays{#3}{#1}%
8291     \advance #4 by #1\relax
8292     \advance #4 by -1373429
8293     \global\bbl@cntcommon=#4\relax}%
8294     #4=\bbl@cntcommon}
8295 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8296     {\countdef\tmpx= 17
8297     \countdef\tmpy= 18
8298     \countdef\tmpz= 19
8299     #6=#3\relax
8300     \global\advance #6 by 3761
8301     \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8302     \tmpz=1 \tmpy=1
8303     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8304     \ifnum \tmpx > #4\relax
8305         \global\advance #6 by -1
8306         \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8307     \fi
8308     \advance #4 by -\tmpx
8309     \advance #4 by 1
8310     #5=#4\relax
8311     \divide #5 by 30
8312     \loop
8313         \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8314         \ifnum \tmpx < #4\relax
8315             \advance #5 by 1
8316             \tmpy=\tmpx
8317     \repeat

```

```

8318 \global\advance #5 by -1
8319 \global\advance #4 by -\tmpy}}
8320 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebyear
8321 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8322 \def\bbl@ca@hebrew#1-#2-#3\@#4#5#6{%
8323 \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8324 \bbl@hebrfromgreg
8325 {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8326 {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebyear}%
8327 \edef#4{\the\bbl@hebyear}%
8328 \edef#5{\the\bbl@hebrmonth}%
8329 \edef#6{\the\bbl@hebrday}}
8330 </ca-hebrew>

```

13.3 Persian

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

```

8331 <*ca-persian>
8332 \ExplSyntaxOn
8333 <<Compute Julian day>>
8334 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8335 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8336 \def\bbl@ca@persian#1-#2-#3\@#4#5#6{%
8337 \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8338 \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8339 \bbl@afterfi\expandafter\gobble
8340 \fi\fi
8341 {\bbl@error{Year~out-of~range}{The~allowed~range~is~2013-2050}}}%
8342 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8343 \ifin@{\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8344 \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8345 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8346 \ifnum\bbl@tempc<\bbl@tempb
8347 \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8348 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8349 \ifin@{\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8350 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8351 \fi
8352 \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8353 \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8354 \edef#5{\fp_eval:n{% set Jalali month
8355 (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8356 \edef#6{\fp_eval:n{% set Jalali day
8357 (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6))}}}%
8358 \ExplSyntaxOff
8359 </ca-persian>

```

13.4 Coptic and Ethiopic

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

```

8360 <*ca-coptic>
8361 \ExplSyntaxOn
8362 <<Compute Julian day>>
8363 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
8364 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8365 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8366 \edef#4{\fp_eval:n{%

```

```

8367 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8368 \edef\bbl@tempc{\fp_eval:n{%
8369 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}}%
8370 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}}%
8371 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}%
8372 \ExplSyntaxOff
8373 </ca-coptic>
8374 *ca-ethiopic>
8375 \ExplSyntaxOn
8376 <<Compute Julian day>>
8377 \def\bbl@ca@ethiopic#1-#2-#3\@@#4#5#6{%
8378 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}}%
8379 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}}%
8380 \edef#4{\fp_eval:n{%
8381 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}}%
8382 \edef\bbl@tempc{\fp_eval:n{%
8383 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}}%
8384 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}}%
8385 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}%
8386 \ExplSyntaxOff
8387 </ca-ethiopic>

```

13.5 Buddhist

That's very simple.

```

8388 *ca-buddhist>
8389 \def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
8390 \edef#4{\number\numexpr#1+543\relax}%
8391 \edef#5{#2}%
8392 \edef#6{#3}}
8393 </ca-buddhist>
8394 %
8395 % \subsection{Chinese}
8396 %
8397 % Brute force, with the Julian day of first day of each month. The
8398 % table has been computed with the help of \textsf{python-lunardate} by
8399 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
8400 % is 2015-2044.
8401 %
8402 % \begin{macrocode}
8403 *ca-chinese>
8404 \ExplSyntaxOn
8405 <<Compute Julian day>>
8406 \def\bbl@ca@chinese#1-#2-#3\@@#4#5#6{%
8407 \edef\bbl@tempd{\fp_eval:n{%
8408 \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}}%
8409 \count@z@
8410 \@tempcnta=2015
8411 \bbl@foreach\bbl@cs@chinese@data{%
8412 \ifnum##1>\bbl@tempd\else
8413 \advance\count@\@ne
8414 \ifnum\count@>12
8415 \count@\@ne
8416 \advance\@tempcnta\@ne\fi
8417 \bbl@xin@{,##1,}{,\bbl@cs@chinese@leap,}%
8418 \ifin@
8419 \advance\count@\m@ne
8420 \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
8421 \else
8422 \edef\bbl@tempe{\the\count@}%
8423 \fi
8424 \edef\bbl@tempb{##1}%
8425 \fi}%

```

```

8426 \edef#4{\the\@tempcnta}%
8427 \edef#5{\bbl@tempe}%
8428 \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
8429 \def\bbl@cs@chinese@leap{%
8430 885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
8431 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
8432 354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
8433 768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
8434 1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
8435 1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
8436 1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
8437 2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
8438 2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
8439 2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
8440 3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
8441 3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
8442 3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
8443 4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
8444 4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
8445 5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
8446 5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
8447 5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
8448 6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
8449 6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
8450 6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
8451 7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
8452 7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
8453 7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
8454 8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8455 8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8456 8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
8457 9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
8458 9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
8459 10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
8460 10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
8461 10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
8462 10896,10926,10956,10986,11015,11045,11074,11103}
8463 \ExplSyntaxOff
8464 \</ca-chinese>

```

14 Support for Plain T_EX (plain.def)

14.1 Not renaming hyphen.tex

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

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

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

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

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

```

8465 (*bplain | blplain)
8466 \catcode`\{=1 % left brace is begin-group character
8467 \catcode`\}=2 % right brace is end-group 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).

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.

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

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.

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 no 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).


```

8503 \def\@gobbletwo#1#2{}
8504 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
8505 \def\@star@or@long#1{%
8506   \@ifstar
8507   {\let\l@ngrel@x\relax#1}%
8508   {\let\l@ngrel@x\long#1}}
8509 \let\l@ngrel@x\relax
8510 \def\@car#1#2\@nil{#1}
8511 \def\@cdr#1#2\@nil{#2}
8512 \let\@typeset@protect\relax
8513 \let\protected@edef\edef
8514 \long\def\@gobble#1{}
8515 \edef\@backslashchar{\expandafter\@gobble\string\}
8516 \def\strip@prefix#1>{}
8517 \def\g@addto@macro#1#2{%
8518   \toks@\expandafter{#1#2}%
8519   \xdef#1{\the\toks@}}
8520 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8521 \def\@nameuse#1{\csname #1\endcsname}
8522 \def\@ifundefined#1{%
8523   \expandafter\ifx\csname#1\endcsname\relax
8524     \expandafter\@firstoftwo
8525   \else
8526     \expandafter\@secondoftwo
8527   \fi}
8528 \def\@expandtwoargs#1#2#3{%
8529   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8530 \def\zap@space#1 #2{%
8531   #1%
8532   \ifx#2\@empty\else\expandafter\zap@space\fi
8533   #2}
8534 \let\bbl@trace\@gobble
8535 \def\bbl@error#1#2{%
8536   \begingroup
8537     \newlinechar=\^^J
8538     \def\{^\^J(babel) }%
8539     \errhelp{#2}\errmessage{\{^\^J}%
8540   \endgroup}
8541 \def\bbl@warning#1{%
8542   \begingroup
8543     \newlinechar=\^^J
8544     \def\{^\^J(babel) }%
8545     \message{\{^\^J}%
8546   \endgroup}
8547 \let\bbl@infowarn\bbl@warning
8548 \def\bbl@info#1{%
8549   \begingroup
8550     \newlinechar=\^^J
8551     \def\{^\^J}%
8552     \wlog{#1}%
8553   \endgroup}

```

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

```

8554 \ifx\@preamblecmds\undefined
8555   \def\@preamblecmds{}
8556 \fi
8557 \def\@onlypreamble#1{%
8558   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8559     \@preamblecmds\do#1}}
8560 \@onlypreamble\@onlypreamble

```

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

```

8561 \def\begindocument{%

```

```

8562 \@begindocumenthook
8563 \global\let\@begindocumenthook\@undefined
8564 \def\do##1{\global\let##1\@undefined}%
8565 \@preamblecmds
8566 \global\let\do\noexpand}

8567 \ifx\@begindocumenthook\@undefined
8568 \def\@begindocumenthook{}
8569 \fi
8570 \@onlypreamble\@begindocumenthook
8571 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

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

```

8572 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
8573 \@onlypreamble\AtEndOfPackage
8574 \def\@endoflfd{}
8575 \@onlypreamble\@endoflfd
8576 \let\bbl@afterlang\@empty
8577 \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.

```

8578 \catcode`\&=\z@
8579 \ifx&\if@filesw\@undefined
8580 \expandafter\let\csname if@filesw\expandafter\endcsname
8581 \csname iffalse\endcsname
8582 \fi
8583 \catcode`\&=4

```

Mimic \LaTeX 's commands to define control sequences.

```

8584 \def\newcommand{\@star@or@long\new@command}
8585 \def\new@command#1{%
8586 \@testopt{\@newcommand#1}0}
8587 \def\@newcommand#1[#2]{%
8588 \ifnextchar [{\@xargdef#1[#2]}%
8589 {\@argdef#1[#2]}}
8590 \long\def\@argdef#1[#2]#3{%
8591 \@yargdef#1\@ne{#2}{#3}}
8592 \long\def\@xargdef#1[#2]#3#4{%
8593 \expandafter\def\expandafter#1\expandafter{%
8594 \expandafter\@protected@testopt\expandafter #1%
8595 \csname\string#1\expandafter\endcsname{#3}}%
8596 \expandafter\@yargdef \csname\string#1\endcsname
8597 \tw@{#2}{#4}}
8598 \long\def\@yargdef#1#2#3{%
8599 \@tempcnta#3\relax
8600 \advance \@tempcnta \@ne
8601 \let\@hash@\relax
8602 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
8603 \@tempcntb #2%
8604 \@whilenum\@tempcntb <\@tempcnta
8605 \do{%
8606 \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
8607 \advance\@tempcntb \@ne}%
8608 \let\@hash@##%
8609 \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
8610 \def\providecommand{\@star@or@long\provide@command}
8611 \def\provide@command#1{%
8612 \begingroup
8613 \escapechar\m@ne\xdef\@gtempa{\string#1}%
8614 \endgroup
8615 \expandafter\ifundefined\@gtempa

```

```

8616     {\def\reserved@a{\new@command#1}}%
8617     {\let\reserved@a\relax
8618     \def\reserved@a{\new@command\reserved@a}}%
8619     \reserved@a}%

8620 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
8621 \def\declare@robustcommand#1{%
8622     \edef\reserved@a{\string#1}%
8623     \def\reserved@b{#1}%
8624     \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
8625     \edef#1{%
8626         \ifx\reserved@a\reserved@b
8627             \noexpand\x@protect
8628             \noexpand#1%
8629         \fi
8630         \noexpand\protect
8631         \expandafter\noexpand\csname
8632             \expandafter\@gobble\string#1 \endcsname
8633     }%
8634     \expandafter\new@command\csname
8635         \expandafter\@gobble\string#1 \endcsname
8636 }
8637 \def\x@protect#1{%
8638     \ifx\protect\@typeset@protect\else
8639         \x@protect#1%
8640     \fi
8641 }
8642 \catcode`\&=\z@ % Trick to hide conditionals
8643 \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`.

```

8644 \def\bbl@tempa{\csname newif\endcsname&fin@}
8645 \catcode`\&=4
8646 \ifx\in@\@undefined
8647     \def\in@#1#2{%
8648         \def\in@@##1##2##3\in@{%
8649             \ifx\in@##2\in@false\else\in@true\fi}%
8650         \in@@#2#1\in@\in@@}
8651 \else
8652     \let\bbl@tempa\@empty
8653 \fi
8654 \bbl@tempa

```

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

```

8655 \def\@ifpackagewith#1#2#3#4{#3}

```

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

```

8656 \def\@ifl@aded#1#2#3#4{}

```

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

```

8657 \ifx\@tempcnta\@undefined
8658     \csname newcount\endcsname\@tempcnta\relax
8659 \fi
8660 \ifx\@tempcntb\@undefined
8661     \csname newcount\endcsname\@tempcntb\relax
8662 \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`).

```

8663 \ifx\bye\undefined
8664   \advance\count10 by -2\relax
8665 \fi
8666 \ifx@ifnextchar\undefined
8667   \def@ifnextchar#1#2#3{%
8668     \let\reserved@#1%
8669     \def\reserved@a{#2}\def\reserved@b{#3}%
8670     \futurelet\@let@token\@ifnch}
8671   \def@ifnch{%
8672     \ifx\@let@token\@sptoken
8673       \let\reserved@c\@xifnch
8674     \else
8675       \ifx\@let@token\reserved@d
8676         \let\reserved@c\reserved@a
8677       \else
8678         \let\reserved@c\reserved@b
8679     \fi
8680   \fi
8681   \reserved@c}
8682 \def\:{\let\@sptoken= }\: % this makes \@sptoken a space token
8683 \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
8684 \fi
8685 \def\@testopt#1#2{%
8686   \@ifnextchar[#{1}{#1[#2]}}
8687 \def\@protected@testopt#1{%
8688   \ifx\protect\@typeset@protect
8689     \expandafter\@testopt
8690   \else
8691     \@x@protect#1%
8692   \fi}
8693 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8694   #2\relax}\fi}
8695 \long\def\@iwhilenum#1{\ifnum #1\relax\expandafter\@iwhilenum
8696   \else\expandafter\@gobble\fi{#1}}

```

14.4 Encoding related macros

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

```

8697 \def\DeclareTextCommand{%
8698   \@dec@text@cmd\providecommand
8699 }
8700 \def\ProvideTextCommand{%
8701   \@dec@text@cmd\providecommand
8702 }
8703 \def\DeclareTextSymbol#1#2#3{%
8704   \@dec@text@cmd\chardef#1{#2}#3\relax
8705 }
8706 \def\@dec@text@cmd#1#2#3{%
8707   \expandafter\def\expandafter#2%
8708     \expandafter{%
8709       \csname#3-cmd\expandafter\endcsname
8710       \expandafter#2%
8711       \csname#3-string#2\endcsname
8712     }%
8713 %   \let@ifdefinable\@rc@ifdefinable
8714   \expandafter#1\csname#3-string#2\endcsname
8715 }
8716 \def\@current@cmd#1{%
8717   \ifx\protect\@typeset@protect\else
8718     \noexpand#1\expandafter\@gobble

```

```

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

```

```

8782 \def\@text@composite#1#2#3\@text@composite{%
8783   \expandafter\@text@composite@x
8784     \csname\string#1-\string#2\endcsname
8785 }
8786 \def\@text@composite@x#1#2{%
8787   \ifx#1\relax
8788     #2%
8789   \else
8790     #1%
8791   \fi
8792 }
8793 %
8794 \def\@strip@args#1:#2-#3\@strip@args{#2}
8795 \def\DeclareTextComposite#1#2#3#4{%
8796   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8797   \bgroup
8798     \lccode`\@=#4%
8799     \lowercase{%
8800   \egroup
8801     \reserved@a @%
8802   }%
8803 }
8804 %
8805 \def\UseTextSymbol#1#2{#2}
8806 \def\UseTextAccent#1#2#3{}
8807 \def\@use@text@encoding#1{}
8808 \def\DeclareTextSymbolDefault#1#2{%
8809   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8810 }
8811 \def\DeclareTextAccentDefault#1#2{%
8812   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8813 }
8814 \def\cf@encoding{OT1}

```

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

```

8815 \DeclareTextAccent{"}{OT1}{127}
8816 \DeclareTextAccent{'}{OT1}{19}
8817 \DeclareTextAccent{^}{OT1}{94}
8818 \DeclareTextAccent{\`}{OT1}{18}
8819 \DeclareTextAccent{\~}{OT1}{126}

```

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

```

8820 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8821 \DeclareTextSymbol{\textquotedblright}{OT1}{`\`}
8822 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
8823 \DeclareTextSymbol{\textquoteright}{OT1}{``'}
8824 \DeclareTextSymbol{\i}{OT1}{16}
8825 \DeclareTextSymbol{\ss}{OT1}{25}

```

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

```

8826 \ifx\scriptsize\@undefined
8827   \let\scriptsize\sevenrm
8828 \fi

```

And a few more “dummy” definitions.

```

8829 \def\language{english}%
8830 \let\bbl@opt@shorthands\@nnil
8831 \def\bbl@ifshorthand#1#2#3{#2}%
8832 \let\bbl@language@opts\@empty
8833 \let\bbl@ensureinfo\@gobble
8834 \let\bbl@provide@locale\relax
8835 \ifx\babeloptionstrings\@undefined

```

```

8836 \let\bbl@opt@strings\@nnil
8837 \else
8838 \let\bbl@opt@strings\babeloptionstrings
8839 \fi
8840 \def\BabelStringsDefault{generic}
8841 \def\bbl@tempa{normal}
8842 \ifx\babeloptionmath\bbl@tempa
8843 \def\bbl@mathnormal{\noexpand\textormath}
8844 \fi
8845 \def\AfterBabelLanguage#1#2{}
8846 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
8847 \let\bbl@afterlang\relax
8848 \def\bbl@opt@safe{BR}
8849 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
8850 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
8851 \expandafter\newif\csname ifbbl@single\endcsname
8852 \chardef\bbl@bidimode\z@
8853 <</Emulate LaTeX>>

A proxy file:
8854 <*plain>
8855 \input babel.def
8856 </plain>

```

15 Acknowledgements

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

References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national \TeX styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The \TeX book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport, *\TeX , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in: \TeX hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018
- [10] Hubert Partl, *German \TeX* , *TUGboat* 9 (1988) #1, p. 70–72.
- [11] Joachim Schrod, *International \TeX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [12] Apostolos Syropoulos, Antonis Tzolomitis and Nick Sofroniu, *Digital typography using \TeX* , Springer, 2002, p. 301–373.
- [13] K.F. Treebus. *Tekstwijzer; een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).