Babel

Localization and internationalization

Unicode
\TeX
pdf\TeX
Lua\TeX
Xe\TeX
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, babel.sty) ........ 8
  3.3 base ........................................ 9
  3.4 key=value 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 Multilanguage strings ...................... 43
  4.11 Macros common to a number of languages ... 49
  4.12 Making glyphs available .................... 49
    4.12.1 Quotation marks ....................... 49
    4.12.2 Letters .................................. 50
    4.12.3 Shorthands for quotation marks ......... 51
    4.12.4 Umlauts and tremas .................... 52
  4.13 Layout ..................................... 53
  4.14 Load engine specific macros ............... 54
  4.15 Creating and modifying languages ........... 54

5 Adjusting the Babel behavior 77
  5.1 Cross referencing macros .................... 79
  5.2 Marks ....................................... 82
  5.3 Preventing clashes with other packages ..... 83
    5.3.1 ifthen .................................. 83
    5.3.2 varioref .................................. 84
    5.3.3 hhline ................................... 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 (babel.def, common) 93

7 Loading hyphenation patterns ................... 94

8 Font handling with fontspec ..................... 98

9 Hooks for XeTeX and LuaTeX 101
  9.1 XeTeX ...................................... 101
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 are 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

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.

```
⟨⟨version=24.1⟩⟩
⟨⟨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.
```

```
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@strip\slash{\expandafter\@gobble\string}
6 \def\bbl@add#1#2{%
7 \bbl@ifunset{\bbl@strip\slash#1}%
8 {\def#1{#2}}%
9 {\expandafter\def\expandafter#1\expandafter{#1#2}}%
10 \def\bbl@isin{\@expandtwoargs\@in}
11 \def\bbl@argc#1{%\expandafter\@expand\@args\@in}
12 \def\bbl@nargc#1{%\expandafter\@expand\@args\@in}
13 \def\bbl@ccarg#1#2#3{%\expandafter\@expand\@args\@in}
14 \expandafter\@expand\@args\@in
15 \def\bbl@carg#1#2{\expandafter#1\csname#2\endcsname%}
16 \def\bbl@c#1{\csname bbl@#1\endcsname%}
17 \def\bbl@cl#1{\csname bbl@#1\languagename\endcsname%}
```

3
\bbladd@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.

\bblafterelse

Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the \else and \fi parts of an if-statement\footnote{This code is based on code presented in TUGboat vol. 12, no2, June 1991 in “An expansion Power Lemma” by Sonja Maus.}. These macros will break if another if\ldots\fi statement appears in one of the arguments and it is not enclosed in braces.

\bbl@exp

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

\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.

\bbl@ifunset

To check if a macro is defined, we create a new macro, which does the same as \texttt{\@ifundefined}. However, in an \textsc{e}-\TeX{} engine, it is based on \texttt{\ifcsname}, which is more efficient, and does not waste
memory. Defined inside a group, to avoid \ifcsname being implicitly set to \relax by the \csname test.

\begin{verbatim}
56 \begingroup
57 \gdef\bbl@ifunset#1{%
58 \expandafter\ifx\csname#1\endcsname\relax
59 \else
60 \expandafter\@firstoftwo
61 \fi}
62 \bbl@ifunset{ifcsname}{}
63 \gdef\bbl@ifunset#1{%
64 \ifcsname#1\endcsname
65 \expandafter\ifx\csname#1\endcsname\relax
66 \bbl@afterelse\expandafter\@firstoftwo
67 \else
68 \bbl@afterfi\expandafter\@secondoftwo
69 \fi}
70 \endgroup
\end{verbatim}

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

\begin{verbatim}
76 \def\bbl@ifblank#1{%
77 \bbl@ifblank@i\nil\nil\nil\@nothing\@nothing\@nothing\@nil
78 \long\def\bbl@ifblank@i#1\nil\nil\nil\@nothing\@nothing\@nothing\@nil{#1}
79 \def\bbl@ifblank@i#1#2#3{%
80 \bbl@ifblank{#1}{}{bl@forkv@eq#1=#2=#3\@nothing}{}
81 \bbl@forkv@eq#1=#2=#3\@nothing{#1}{#2}{#3}
82 \def\bbl@forkv#1#2{%
83 \def\bbl@forcmd##1{#2}%
84 \bbl@fornext#1,
85 \ifx\@nothing#1elax\else
86 \bbl@ifblank{#1}{}{bl@trim\bbl@forcmd{#1}}%
87 \expandafter\bbl@fornext
88 \fi}
89 \def\bbl@forkv@eq#1=#2=#3#4{%
90 \bbl@trim\bbl@forcmd#2#3\@nothing\@nothing
91 \bbl@forkv@eq#1=#2=#3#4%
92 \bbl@forkv@eq#1=#2=#3#4{
93 \bbl@forkv@eq#1=#2=#3#4{
94 \bbl@forkv@eq#1=#2=#3#4{
95 \bbl@forkv@eq#1=#2=#3#4{
96 \bbl@forkv@eq#1=#2=#3#4{
97 \bbl@forkv@eq#1=#2=#3#4{
98 \bbl@forkv@eq#1=#2=#3#4{
99 \bbl@forkv@eq#1=#2=#3#4{
100 \bbl@forkv@eq#1=#2=#3#4{
A for loop. Each item (trimmed) is #1. It cannot be nested (it’s doable, but we don’t need it).

101 \def\bbl@vforeach#1#2{%
102 \def\bbl@forcmd#1#2#3{#2}
103 \bbl@forcmd#1#2#3#4{#2}{#3}{#4}
\end{verbatim}

\bbl@replace Returns implicitly \toks@ with the modified string.

\begin{verbatim}
104 \def\bbl@replace#1#2#3{#1 -> repl #2 by #3
105 \toks@{#3}
106 \def\bbl@replace@aux#1#2#3#4{#1}
\end{verbatim}

\endinput
An extension to the previous macro. It takes into account the parameters, and it is string based (i.e., if you replace \relax by ho, then \rho 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 \textbf{\texttt{\@date}}, and also fails if there are macros with spaces, because they are retokenized). It may change! (or even merged with \textbf{\texttt{\@replace}}; I'm not sure checking the replacement is really necessary or just paranoia).

Two further tools. \textbf{\texttt{\@ifsamestring}} first expand its arguments and then compare their expansion (sanitized, so that the catcodes do not matter). \textbf{\texttt{\@engine}} takes the following values: 0 is pdf\TeX, 1 is luatex, and 2 is xetex. You may use the latter in your language style if you want.
A somewhat hackish tool (hence its name) to avoid spurious spaces in some contexts.

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.

The following adds some code to \extras... both before and after, while avoiding doing it twice. It's somewhat convoluted, to deal with #

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

\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 requires loading switch.def in the format.

3.1 Multiple languages
Another counter is used to keep track of the allocated languages. \TeX{} and \LaTeX{} reserves for this purpose the count 19.

This macro was introduced for \TeX{} < 2. Preserved for compatibility.

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)

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

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

```latex
\def\AfterBabelLanguage#1{% 
\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.

```latex
\ifx\bbl@languages\undefined\else
\begingroup
\catcode`^^I=12
\@ifpackagewith{babel}{showlanguages}{%
\begingroup
\def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
\wlog{<*languages>}%
\bbl@languages%
\wlog{</languages>}%
\endgroup}{%}
\endgroup
\def\bbl@elt#1#2#3#4{% 
\ifnum#2=\z@ 
\gdef\bbl@nulllanguage{#1}%
\def\bbl@elt##1##2##3##4{}%
\fi}
\bbl@languages%
\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 EPiX 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.
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 \BabelOptions 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@{=}{#1}%
303 \ifin@
304 \edef\bbl@tempc{\bbl@tempc\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
305 \else
306 \in@{\$modifiers\$}{#1}$% TODO. Allow spaces.
307 \ifin@
308 \bbl@tempe#2\@@
309 \else
310 \in@[\={#1}]
311 \ifin@
312 \edef\bbl@tempc{\bbl@tempc\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
313 \else
314 \edef\bbl@tempc{\bbl@tempc\ifx\bbl@tempc\@empty\else\bbl@tempc\fi\#1}%
315 \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
316 \bbl@tempe#2\@@
317 \else
318 \fi
319 \fi
320 \fi
321 \let\bbl@tempc@empty
322 \bbl@foreach\bbl@tempa{\bbl@tempd#1.@\@\@nil}
323 \expandafter\let\csname opt@babel.sty\endcsname\bbl@tempc

Thenext 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=\% main -> +1]{provide=*}{\chardef\bbl@iniflag\@ne}
335 \chardef\bbl@iniflag[2]{\chardef\bbl@iniflag\tw@}{ % add = 2
336 \chardef\bbl@iniflag[3]{\chardef\bbl@iniflag\thr@@} % add + main
337 % A separate option
338 \DeclareOption[provide=\% main -> +1\% main -> +1\% main -> +1\% main -> +1]{provide=+}{\chardef\bbl@iniflag\z@
339 \chardef\bbl@load@options@empty\bbl@load@options\{\chardef\bbl@autoload@options\import
340 % Don't use. Experimental. TODO.
341 \newif\iff@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 have 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.

\let\bbl@opt@shorthands\@nnil
\let\bbl@opt@config\@nnil
\let\bbl@opt@main\@nnil
\let\bbl@opt@headfoot\@nnil
\let\bbl@opt@layout\@nnil
\let\bbl@opt@provide\@nnil

The following tool is defined temporarily to store the values of options.
\def\bbl@tempa#1=#2\bbl@tempa{\%\ifx{opt@#1}\@nnil\bbl@csarg\edef{opt@#1}{#2}\%\else\bbl@error\{Bad option '#1=#2'. Either you have misspelled the\% key or there is a previous setting of ‘#1’. Valid\% keys are, among others, ‘shorthands’, ‘main’, ‘bidi’,\% ‘strings’, ‘config’, ‘headfoot’, ‘safe’, ‘math’.)\% \{See the manual for further details.\}\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.

\let\bbl@language@opts\@empty
\DeclareOption*{% \begin{verbatim}
\def\bbl@tempa#1=#2\bbl@tempa{\%\ifx{opt@#1}\@nnil\bbl@csarg\edef{opt@#1}{#2}\%\else\bbl@error\{Bad option ‘#1=’\%. Either you have misspelled the\% key or there is a previous setting of ‘#1’. Valid\% keys are, among others, ‘shorthands’, ‘main’, ‘bidi’,\% ‘strings’, ‘config’, ‘headfoot’, ‘safe’, ‘math’.)\% \{See the manual for further details.\}\fi}
\end{verbatim} %

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

\ProcessOptions*\%
\iffalse\bbl@opt@provide\@nnil\else\chardef\bbl@iniflag\@ne\bbl@exp{\\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{% \in@{,provide,}{,#1,} %\ifin@\def\bbl@opt@provide{#2} %\bbl@replace\bbl@opt@provide{;}{,} %\fi} %\fi\%

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=....

\bbl@trace{Conditional loading of shorthands} \def\bbl@sh@string#1{\% \iffalse\empty\else \\iffalse\string-\% \else\iffalse\string\,\% \else\string\% \fi} \%
\newpage

11
The following macro tests if a shorthand is one of the allowed ones.

\def\bbl@ifshorthand#1{%  
  \bbl@xin{@string#1}{\bbl@opt@shorthands}%  
  \ifin@  
  \expandafter\@firstoftwo  
  \else  
  \expandafter\@secondoftwo  
  \fi}

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

\edef\bbl@opt@shorthands{%  \expandafter\bbl@sh@string\bbl@opt@shorthands\@empty}%

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

\bbl@ifshorthand{'}%  \PassOptionsToPackage{activeacute}{babel}}{}  \bbl@ifshorthand{`}%  \PassOptionsToPackage{activegrave}{babel}}{}

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.

\g@addto@macro\resetactivechars{%  \set@typeset@protect  \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%  \let\protect\noexpand}%

For the option safe we use a different approach -- \bbl@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.

\def\bbl@opt@safe{BR}%

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

\bbl@trace{Defining IfBabelLayout}
\newcommand\IfBabelLayout[3]{\if\bbl@exp{\ifie\bbl@opt@layout\ifie\nameuse{\bbl@opt@babel.sty}}{\expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}}{\bbl@opt@layout}{\#1}{\#3}
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*.

\fi % Same line!
\ProvidesFile{babel.def}[\version] Babel common definitions
\fi
\ProvidesFile{babel.def}[\date] v\version Babel common definitions
\fi

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

4 Multiple languages

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

\def\bbl@version{
\version
}\def\bbl@date{
\date
}\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.
\def\adddialect#1#2{\global\chardef#1#2} \begin{group}
\count@#1\relax
\adddialect{adddialect}{(#1)(#2)}
\begingroup
\count@#1\relax
\def\bbl@elt##1##2##3##4{\ifnum\count@=##2\relax\edef\bbl@tempa{\expandafter\@gobbletwo\string#1}\bbl@info{Hyphen rules for '{\expandafter\string\csname l@##1\endcsname}'\%\string\language\the\count@. Reported}\fi}
\bbl@cs{languages}\endgroup
\bbl@iflanguage executes code only if the language \language exists. Otherwise raises an error. The argument of \bbl@iflanguage 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 \MakeXXX case. 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 \language is encapsulated, so that its case does not change.
\def\bbl@fixname#1{\begin{group}
\def\bbl@fixname#1{\begin{group}
\def\bbl@tempe{\endgroup}
\endgroup}
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-latin-ba becomes fr-Latn-BA. Note #4 may contain some \@empty’s, but they are eventually removed. \bbl@bcplookup either returns the found ini or it is \relax.

\def\bbl@bcpcase#1#2#3#4\@@#5{% 
\ifx\@empty#3%
\uppercase{\def#5{#1#2}}%
\else
\uppercase{\def#5{#1}}%
\lowercase{\edef#5{#5#2#3#4}}%
\fi
\def\bbl@bcplookup#1-#2-#3-#4\@@{%
\let\bbl@bcp\relax
\lowercase{\def\bbl@tempa{#1}}%
\ifx\@empty#2%
\IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
\else\ifx\@empty#3%
\bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
\IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
{\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
{}%
\ifx\bbl@bcp\relax
\IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}
\fi
\else
\bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
\bbl@bcpcase#3\@empty\@empty\@@\bbl@tempc
\IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
{\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
{}%
\ifx\bbl@bcp\relax
\IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}{}
\fi
\fi
\fi
\ifx\bbl@bcp\relax
\IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}{}
\fi
\fi}
\let\bbl@initoload\relax
⟨-core⟩
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.

4.1 Selecting the language

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.

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.

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

\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 \texttt{T\LaTeX}'s \texttt{aftergroup} mechanism to help us. The command \texttt{aftergroup} stores the token immediately following it to be executed when the current group is closed. So we define a temporary control sequence \texttt{bbl@pop@language} to be executed at the end of the group. It calls \texttt{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 \texttt{bbl@language@stack} and initially empty:

\begin{verbatim}
\def\bbl@language@stack{}
\end{verbatim}

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:

\begin{verbatim}
\def\bbl@push@language{%
  \ifx\languagename\@undefined\else
    \ifx\currentgrouplevel\@undefined
      \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
    \else
      \ifnum\currentgrouplevel=\z@\xdef\bbl@language@stack{\languagename+}\fi
    \else
      \xdef\bbl@language@stack{\languagename+\bbl@language@stack}\fi
  \fi
}\end{verbatim}

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 \texttt{\languagename}. For this we first define a helper function.

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

\begin{verbatim}
\def\bbl@pop@lang#1+#2\@@{%
  \edef\languagename{#1}
  \xdef\bbl@language@stack{#2}}
\end{verbatim}

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 \texttt{bbl@pop@lang} is executed \texttt{T\LaTeX} first expands the stack, stored in \texttt{\bbl@language@stack}. The result of that is that the argument string of \texttt{bbl@pop@lang} contains one or more languagenames, each followed by a '+'-sign (zero languagenames won't occur as this macro will only be called after something has been pushed on the stack).

\begin{verbatim}
\let\bbl@ifrestoring\@secondoftwo
\def\bbl@push@language{%
  \\ifx\languagename\@undefined\else
    \\ifx\currentgrouplevel\@undefined
      \xdef\bbl@language@stack{\languagename+\bbl@language@stack}\fi
    \else
      \\ifnum\currentgrouplevel=\z@\xdef\bbl@language@stack{\languagename+}\fi
    \else
      \xdef\bbl@language@stack{\languagename+\bbl@language@stack}\fi
  \fi
}\end{verbatim}

Once the name of the previous language is retrieved from the stack, it is fed to \texttt{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 \texttt{\localeid}. This means \texttt{\localeid}... will be reserved for hyphenation patterns (so that two locales can share the same rules).

\begin{verbatim}
\chardef\localeid\z@
\def\bbl@id@assign{\def\bbl@id@assign{%
  \ifunset{bbl@id@@\languagename}{}
  \count@\bbl@id@last\relax
\end{verbatim}

16
The unprotected part of \selectlanguage.

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

We also write a command to change the current language in the auxiliary files. \bbl@savelastskip is used to deal with skips before the write whatever (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).
First, check if the user asks for a known language. If so, update the value of \language and call \originalTeX to bring \TeX in a certain pre-defined state.

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

Then we have to redefine \originalTeX to compensate for the things that have been activated. To save memory space for the macro definition of \originalTeX, we construct the control sequence name for the \noextras⟨lang⟩ command at definition time by expanding the \csname primitive.

Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of \selectlanguage, and calling these macros.

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

\def\bbl@switch#1{% from select@, foreign@
  \bbl@ensureinfo{#1}%
  \originalTeX{noextras#1}{\let\originalTeX\@empty\babel@beginsave}{\bbl@usehooks{afterreset}{}}
  \languageshorthands{none}{\selectlanguage{#1}}
}

\newif\ifbbl@usedategroup
\let\bbl@savedextras\@empty
\def\bbl@switch#1{% from select@, foreign@
  \bbl@ensureinfo{#1}%
  \originalTeX{noextras#1}{\let\originalTeX\@empty\babel@beginsave}{\bbl@usehooks{afterreset}{}}
  \languageshorthands{none}{\selectlanguage{#1}}
}

\newif\ifbbl@bcpallowed
\let\bbl@savelastskip\relax
\let\bbl@restorelastskip\relax
\newif\ifbbl@bcpallowed
\bbl@bcpallowedfalse
\def\select@language#1{% from set@, babel@aux
  \ifx\bbl@selectorname\@empty
    \def\bbl@selectorname{select}%
  \fi
  \ifnum\bbl@hymapsel=\@cclv
    \chardef\bbl@hymapsel4\relax
  \fi
  \edef\languagename{#1}%
  \bbl@fixname\languagename
  \bbl@provide@locale
  \bbl@iflanguage\languagename{% 
    \let\bbl@select@type\z@
    \expandafter\bbl@switch\expandafter{\languagename}}
}

\def\babel@aux#1#2{% 
  \select@language{#1}%
  \bbl@foreach\BabelContentsFiles{% 
    \@writefile{##1}{\babel@toc{#1}{#2}\relax}}%
  \def\babel@toc#1#2{%
  \select@language{#1}}
\bbl@id@assign
% switch captions, date
\bbl@bsphack
\ifcase\bbl@select@type
  \csname captions#1\endcsname\relax
  \csname date#1\endcsname\relax
\else
  \bbl@xin{,captions,}{,\bbl@select@opts,}%
  \ifin@
    \csname captions#1\endcsname\relax
  \fi
  \bbl@xin{,date,}{,\bbl@select@opts,}%
  \ifin@ % if \foreign... within <lang>date
    \csname date#1\endcsname\relax
  \fi
\fi
\bbl@esphack
% switch extras
\csname bbl@preextras@#1\endcsname
\bbl@usehooks{beforeextras}{}%
\csname extras#1\endcsname\relax
\bbl@usehooks{afterextras}{}%
% > babel-ensure
% > babel-sh-<short>
% > babel-bidi
% > babel-fontspec
\let\bbl@savedextras\@empty
% hyphenation - case mapping
\ifcase\bbl@opt@hyphenmap\or
  \def\BabelLower##1##2{\lccode##1=##2\relax}%
  \ifnum\bbl@hymapsel>4\else
    \csname\languagename @bbl@hyphenmap\endcsname
  \fi
\chardef\bbl@opt@hyphenmap\@z@
\else
  \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
    \csname\languagename @bbl@hyphenmap\endcsname
  \fi
\fi
\let\bbl@hymapsel\@cclv
% hyphenation - select rules
\ifnum\csname l@\languagename\endcsname=\l@unhyphenated
  \edef\bbl@tempa{u}%
\else
  \edef\bbl@tempa{%
bbl@tempa{}\bbl@cl{l
brk}}%
\fi
% linebreaking - handle u, e, k (v in the future)
\bbl@xin{u}{/\bbl@tempa}%
  \ifin@\else\bbl@xin{e}{/\bbl@tempa}\fi % elongated forms
  \ifin@\else\bbl@xin{k}{/\bbl@tempa}\fi % only kashida
  \ifin@\else\bbl@xin{p}{/\bbl@tempa}\fi % padding (eg, Tibetan)
  \ifin@\else\bbl@xin{v}{/\bbl@tempa}\fi % variable font
\fi
% unhyphenated/kashida/elongated/padding = allow stretching
\language\@unhyphenated
\bbl@savevariable\emergencystretch
\emergencystretch\maxdimen
\bbl@savevariable\hbadness
\hbadness\@M
\else
  % other = select patterns
  \bbl@patterns(#1)%
\fi
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.

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 \foreignlanguage.

\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.

\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.

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.

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

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 babel@hyphenation, 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.

\let\bbl@hyphlist\@empty
\let\bbl@hyphenation\relax
\let\bbl@pttnlist\@empty
\let\bbl@patterns\relax
\let\bbl@hymapsel=\@cclv
\def\bbl@patterns#1{%
  \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
  \csname l@#1\endcsname
  \edef\bbl@tempa{#1}%
  \else
  \csname l@#1:\f@encoding\endcsname
  \edef\bbl@tempa{#1:\f@encoding}%
  \fi
% > luatex
\@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
% % > latex
\@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
  \begingroup
  \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
  \ifin@\else
    \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
    \hyphenation{\bbl@hyphenation@
    \@ifundefined{bbl@hyphenation@#1}{}{\space\csname bbl@hyphenation@#1\endcsname}}%
    \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
    \fi
  \endgroup}
\def\hyphenrules#1{%
  \edef\bbl@tempf{#1}%
  \bbl@fixname\bbl@tempf
  \bbl@iflanguage\bbl@tempf{%
    \expandafter\expandafter\expandafter\bbl@patterns\expandafter{\bbl@tempf}%
    \ifx\languageshorthands\@undefined\else
      \languageshorthands{none}%
    \fi
    \expandafter\expandafter\expandafter\set@hyphenmins
    \csname\bbl@tempf hyphenmins\endcsname\relax%}
    \else
      \set@hyphenmins\tw@\thr@@\relax%}
    \fi
  \endgroup}
\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 \langle lang\rangle hyphenmins is already defined this command has no effect.

\def\providehyphenmins#1#2{%
  \expandafter\ifx\csname #1hyphenmins\endcsname\relax
    \@namedef{#1hyphenmins}[#2]%
  \fi}
\set@hyphenmins
This macro sets the values of \lefthyphenmin and \righthyphenmin. It expects two values as its argument.

\def\set@hyphenmins#1#2{%
The identification code for each file is something that was introduced in \LaTeX2\epsilon. When the command \ProvidesFile does not exist, a dummy definition is provided temporarily. For use in the language definition file the command \ProvidesLanguage is defined by babel. Depending on the format, i.e., on if the former is defined, we use a similar definition or not.

\ProvidesLanguage\ife\ProvidesFile\@undefined\def\ProvidesLanguage#1\[#2 #3 #4\]{\wlog{Language: #1 #4 #3 <#2>}\}
\else\def\ProvidesLanguage#1{\begingroup\catcode`\ 10 \%\@makeother/\%
\@ifnextchar\[%\]{{\@provideslanguage{#1}}}{{\@provideslanguage{#1}\[]}}\endgroup}\fi

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

\providecommand\setlocale{}
\let\bbl@error\bbl@error
\let\uselocale\setlocale
\let\locale\setlocale
\let\selectlocale\setlocale
\let\textlocale\setlocale
\let\textlanguage\setlocale
\let\languagetext\setlocale

\section{Errors}

\nolerr\nopatterns\  The babel package will signal an error when a document 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 \USenglish, but it might also be empty.

\nooterr\  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 \TeX, 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.
\edef\bbl@nulllanguage{\string\language=0}\def\bbl@nocaption{\protect\bbl@nocaption@i}\def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname\global\namedef{#2}{\textbf{?#1?}}\@nameuse{#2}}
Here ended the now discarded switch.def. Here also (currently) ends the base option.

\def\bbl@trace{Compatibility with language.def}
\ifx\bbl@languages\@undefined
\input luababel.def
\else
\fi
\fi
\bbl@trace{Compatibility with language.def}
\ifx\bbl@languages\@undefined
\message{I could not find the file language.def}
\else
\message{I could not find the file language.def}
\begin{group}
\def\addlanguage{#1#2#3#4#5}{%
\expandafter\ifx\csname lang@#1\endcsname\relax\else
\global\expandafter\let\csname l@#1\expandafter\endcsname
\expandafter\csname lang@#1\endcsname
\fi}%
\def\uselanguage{}%
\input language.def
\endgroup
\fi
\fi
\def\chardef\lenglish{\@english}
\fi
\addto It takes two arguments, a ⟨control sequence⟩ and Tex-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.

\begin{verbatim}
def\addto#1#2{\ifx#1\@undefined\def#1{#2}\else\ifx#1\relax\def#1{#2}\else\{\toks@\expandafter{#1#2}\}\edef#1{\the	oks@}\fi\fi}
def\bbl@withactive#1#2{\begingroup\lccode`~=`#2\relax\lowercase{\endgroup#1~}}
\def\bbl@redefine#1{\edef\bbl@tempa{\bbl@stripslash#1}\expandafter\let\csname org@\bbl@tempa\endcsname#1\expandafter\def\csname\bbl@tempa\endcsname}
\def\bbl@redefine@long#1{\edef\bbl@tempa{\bbl@stripslash#1}\bbl@ifunset{\bbl@tempa\space}\expandafter\let\csname org@\bbl@tempa\endcsname\def\bbl@tempa{\bbl@tempa\endcsname}\expandafter\let\csname org@\bbl@tempa\endcsname\def\bbl@tempa{\bbl@tempa\endcsname}\@namedef{\bbl@tempa\space}}\bbl@useshooks
\def\bbl@withactive#1#2{\begingroup\lccode`~=`#2\relax\lowercase{\endgroup#1~}}
\def\bbl@redefine#1{\edef\bbl@tempa{\bbl@stripslash#1}\expandafter\let\csname org@\bbl@tempa\endcsname#1\expandafter\def\csname\bbl@tempa\endcsname}
\def\bbl@redefine@long#1{\edef\bbl@tempa{\bbl@stripslash#1}\bbl@ifunset{\bbl@tempa\space}\expandafter\let\csname org@\bbl@tempa\endcsname\def\bbl@tempa{\bbl@tempa\endcsname}\expandafter\let\csname org@\bbl@tempa\endcsname\def\bbl@tempa{\bbl@tempa\endcsname}\@namedef{\bbl@tempa\space}}\bbl@useshooks
\def\bbl@withactive#1#2{\begingroup\lccode`~=`#2\relax\lowercase{\endgroup#1~}}
\def\bbl@redefine#1{\edef\bbl@tempa{\bbl@stripslash#1}\expandafter\let\csname org@\bbl@tempa\endcsname#1\expandafter\def\csname\bbl@tempa\endcsname}
\def\bbl@redefine@long#1{\edef\bbl@tempa{\bbl@stripslash#1}\bbl@ifunset{\bbl@tempa\space}\expandafter\let\csname org@\bbl@tempa\endcsname\def\bbl@tempa{\bbl@tempa\endcsname}\expandafter\let\csname org@\bbl@tempa\endcsname\def\bbl@tempa{\bbl@tempa\endcsname}\@namedef{\bbl@tempa\space}}\bbl@useshooks
\end{verbatim}

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.

\begin{verbatim}
def\bbl@withactive#1#2{\begingroup\lccode`~=`#2\relax\lowercase{\endgroup#1~}}
def\bbl@redefine#1{\edef\bbl@tempa{\bbl@stripslash#1}\expandafter\let\csname org@\bbl@tempa\endcsname#1\expandafter\def\csname\bbl@tempa\endcsname}
def\bbl@redefine@long#1{\edef\bbl@tempa{\bbl@stripslash#1}\bbl@ifunset{\bbl@tempa\space}\expandafter\let\csname org@\bbl@tempa\endcsname\def\bbl@tempa{\bbl@tempa\endcsname}\expandafter\let\csname org@\bbl@tempa\endcsname\def\bbl@tempa{\bbl@tempa\endcsname}\@namedef{\bbl@tempa\space}}
def\bbl@redefinerobust#1{\edef\bbl@tempa{\bbl@stripslash#1}\bbl@ifunset{\bbl@tempa\space}\expandafter\let\csname org@\bbl@tempa\endcsname\def\bbl@tempa{\bbl@tempa\endcsname}\expandafter\let\csname org@\bbl@tempa\endcsname\def\bbl@tempa{\bbl@tempa\endcsname}\@namedef{\bbl@tempa\space}}
\end{verbatim}

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@useshooks is the commands used by babel to execute hooks defined for an event.

\begin{verbatim}
\def\bbl@trace{Hooks}
\def\bbl@ifunset{bbl\@hk@#2}{\EnableBabelHook[#2]}{}
\end{verbatim}
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 `\@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.
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 at-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
When \texttt{#2} was not a control sequence we construct one and compare it with \texttt{\relax}.

Finally we check \texttt{\originalTeX}.

\begin{verbatim}
\def\bbl@ldfinit{\let\bbl@screset\@empty
\let\BabelStrings\bbl@opt@string
\let\BabelOptions\@empty
\let\BabelLanguages\relax
\ifx\originalTeX\@undefined
\let\originalTeX\@empty
\else
\originalTeX
\fi}
\def\LdfInit#1#2{\chardef\atcatcode=\catcode\@
\catcode\@=11\relax
\chardef\eqcatcode=\catcode\=
\catcode\==12\relax
\expandafter\if\expandafter\@backslashchar
\expandafter\@car\string#2\@nil
\ifx#2\@undefined\else
\ldf@quit{#1}\
\fi
\else
\expandafter\ifx\csname#2\endcsname\relax\else
\ldf@quit{#1}\
\fi
\fi
\bbl@ldfinit}
\ldf@quit This macro interrupts the processing of a language definition file.
\begin{verbatim}
\def\ldf@quit#1{\expandafter\main@language\expandafter{#1}\
\catcode\@=\atcatcode \let\atcatcode\relax
\catcode\==\eqcatcode \let\eqcatcode\relax
\endinput}
\end{verbatim}
\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.

\begin{verbatim}
\def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
\bbl@afterlang
\let\BabelLanguages\relax
\let\Bbl@screset\relax
\def\ldf@finish#1{\loadlocalcfg{#1}\
\bbl@afterldf{#1}\
\expandafter\main@language\expandafter{#1}\
\catcode\@=\atcatcode \let\atcatcode\relax
\catcode\==\eqcatcode \let\eqcatcode\relax
\end{verbatim}
\end{verbatim}

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

\begin{verbatim}
\onlypreamble\LdfInit
\onlypreamble\ldf@quit
\onlypreamble\ldf@finish
\end{verbatim}

28
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.

\begin{verbatim}
\def\main@language#1{\def\bbl@main@language{#1}\let\languagename\bbl@main@language % TODO. Set localename\bbl@id@assign\bbl@patterns{\languagename}}
\end{verbatim}

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

\begin{verbatim}
\def\bbl@beforestart{\def\@nolanerr##1{\bbl@warning{Undefined language '##1' in aux. \Reported}}}\bbl@usehooks{beforestart}{% Group!\if@filesw\providecommand\babel@aux[2]{}% \immediate\write\@mainaux{\string\providecommand\string\babel@aux[2]{}}\immediate\write\@mainaux{\string\@nameuse{\bbl@beforestart}}\fi\expandafter\selectlanguage\expandafter{\bbl@main@language}⟨-core⟩\if\bbl@single % must go after the line above.\renewcommand\selectlanguage[1]{}\renewcommand\foreignlanguage[2]{#2}\global\let\babel@aux\@gobbletwo % Also as flag\fi\end{verbatim}

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

\begin{verbatim}
\def\select@language@x#1{\ifcase\bbl@select@type\bbl@ifsamestring\languagename{#1}{}{\select@language{#1}}\else\select@language{#1}\fi}
\end{verbatim}

\section{Shorthands}

The macro \texttt{\bbl@add@special} is used to add a new character (or single character control sequence) to the macro \texttt{\dospecials} (and \texttt{@sanitize} if \LaTeX{} 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 \fss@catcodes, added in 3.10.

\bbl@trace{Shorthand}
\def\bbl@add@special#1{% 1:a macro like ", ?, etc.
  \bbl@dospecials{do#1}% test @sanitize = \relax, for back. compat.
  \ifx\fss@catcodes\undefined\else % TODO - same for above
  \begingroup
    \catcode`#1\active
    \fss@catcodes
    \ifnum\catcode`#1=\active
      \endgroup
    \bbl@add\fss@catcodes{\@makeother#1}\
  \else
    \endgroup
  \fi
\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.

\def\bbl@remove@special#1{% 1:\endgroup \edef\noexpand\dospecials{\dospecials}\
  \expandafter\ifx\csname @sanitize\endcsname\relax\else
    \def\noexpand\@sanitize{\@sanitize}\fi
\}\
\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, \langle level\rangle@group, \langle level\rangle@active and \langle next-level\rangle@active (except in system).

\def\bbl@active@def#1#2#3#4{% 1:\n@namedef{#3#1}{\n  \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
    \bbl@afterelse\bbl@sh@select#2#1{#3|#arg1}{#4#1}\n  \else
    \bbl@afterfi\csname#2@sh@#1@\endcsname
  \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.
\long\@namedef{#3@arg#1}##1{% 
ex{csname#2@sh@#1@string#2@1}\endcsname\relax 
bb1@afterelse\csname#2@1\endcsname#1% 
else 
bb1@afterfi\csname#2@sh@1@string#2@1\endcsname 
\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.
\def\initiate@active@char#1{% 
bbl@ifunset{active@char\string#1}{} 
\bbl@withactive{\expandafter\@initiate@active@char\expandafter}#1\string#1#1 
{} 
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).
\def\@initiate@active@char#1#2#3{% 
bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax} 
@ifx#1\@undefined 
bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}} 
\else 
\bbl@csarg\let{oridef@@#2}#1\bbl@csarg\edef{oridef@#2}{\let
oexpand#1\expandafter\noexpand\csname bbl@oridef@@#2\endcsname} 
\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 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").
\ifx#1#3\relax 
\expandafter\let\csname normal@char#2\endcsname#3{} 
\else 
\bbl@info{Making #2 an active character}% \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi 
\@namedef{normal@char#2}{\textormath{#3}{\csname bbl@oridef@@#2\endcsname}} 
\else 
\@namedef{normal@char#2}{#3} 
\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).
\bbl@restoreactive{#2} 
\AtBeginDocument{% 
}\catcode`#2\active 
\if@filesw 
\immediate\write\@mainaux{\catcode`\string#2\active}% 
\fi 
\expandafter\bbl@add@special\csname#2\endcsname 
\catcode`#2\active 
\fi 
Now we have set \normal@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)).

\let\bbl@tempa\@firstoftwo
\if\string^#2%\def\bbl@tempa{\noexpand\textormath}%\else\ifx\bbl@mathnormal\@undefined\else\let\bbl@tempa\bbl@mathnormal\fi\fi\expandafter\edef\csname active@char#2\endcsname{\bbl@tempa{\noexpand\if@safe@actives\noexpand\expandafter\csname normal@char#2\endcsname\noexpand\else\noexpand\expandafter\csname bbl@doactive#2\endcsname\noexpand\fi}}%\bbl@csarg\edef{doactive#2}{%\expandafter\noexpand\csname user@active#2\endcsname}%

Wenowdefine 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!).

\bbl@csarg\edef{active@#2}{%\noexpand\active@prefix\noexpand#1%\expandafter\noexpand\csname active@char#2\endcsname}%
\bbl@csarg\edef{normal@#2}{%\noexpand\active@prefix\noexpand#1%\expandafter\noexpand\csname normal@char#2\endcsname}%
\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.

\bbl@active@def#2\user@group{user@active}{language@active}%
\bbl@active@def#2\language@group{language@active}{system@active}%
\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` \protect. To prevent this from happening a couple of shorthand needs to be defined at user level.

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

\if\string'#2%\let\prim@\bbl@prim@s\let\active@math@prime#1%\fi

\bbl@usehooks{initiateactive}{(#1)(#2)(#3)}

32
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 \@ifpackagewith{babel}{KeepShorthandsActive}{}
1320 \@ifpackagewith{babel}{KeepShorthandsActive}{}
1321 \@ifpackagewith{babel}{KeepShorthandsActive}{}
1322 \@ifpackagewith{babel}{KeepShorthandsActive}{}
1323 \@ifpackagewith{babel}{KeepShorthandsActive}{}
1324 \@ifpackagewith{babel}{KeepShorthandsActive}{}
1325 \@ifpackagewith{babel}{KeepShorthandsActive}{}
1326 \@ifpackagewith{babel}{KeepShorthandsActive}{}

\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{\expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1328 \bbl@afterelse\bbl@firstcs
1329 \bbl@afterelse\bbl@scndcs
1330 \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1331 \bbl@afterfi}

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

1333 \begingroup
1334 \bbl@ifunset{ifincsname}{} {\def\active@prefix#1{\ifx\csname#1@sh@#2@sel\endcsname\relax
1335 \bbl@afterelse\bbl@firstcs
1336 \bbl@afterelse\bbl@scndcs
1337 \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1338 \bbl@afterfi}}
1339 \bbl@ifunset{ifincsname}{} {\def\active@prefix#1{\ifx\csname#1@sh@#2@sel\endcsname\relax
1340 \bbl@afterelse\bbl@firstcs
1341 \bbl@afterelse\bbl@scndcs
1342 \bbl@afterfi\bbl@afterelse\@gobble
1343 \bbl@afterfi}}
1344 \bbl@ifunset{ifincsname}{} {\def\active@prefix#1{\ifx\csname#1@sh@#2@sel\endcsname\relax
1345 \bbl@afterelse\bbl@firstcs
1346 \bbl@afterelse\bbl@scndcs
1347 \bbl@afterfi\bbl@afterelse\@gobble
1348 \bbl@afterfi}}
1349 \bbl@ifunset{ifincsname}{} {\def\active@prefix#1{\ifx\csname#1@sh@#2@sel\endcsname\relax
1350 \bbl@afterelse\bbl@firstcs
1351 \bbl@afterelse\bbl@scndcs
1352 \bbl@afterfi\bbl@afterelse\@gobble
1353 \bbl@afterfi}}
1354 \bbl@ifunset{ifincsname}{} {\def\active@prefix#1{\ifx\csname#1@sh@#2@sel\endcsname\relax
1355 \bbl@afterelse\bbl@firstcs
1356 \bbl@afterelse\bbl@scndcs
1357 \bbl@afterfi\bbl@afterelse\@gobble
1358 \bbl@afterfi}}

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\langle\char\rangle. When this expansion mode is active (with \@safe@actives true), something like "13" becomes \texttt{12} in an \edef (in other words, shortcodes are \texttt{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@actives false).

When this expansion mode is active (with \@safe@actives true), something like "13" becomes \texttt{12} in an \edef (in other words, shortcodes are \texttt{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@actives false).

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.

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\langle\char\rangle in the case of \bbl@activate, or \normal@char\langle\char\rangle in the case of \bbl@deactivate.

These macros are used only as a trick when declaring shorthands.

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 \LaTeX\ code in text mode, (2) the string for hyperref, (3) the \LaTeX\ 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.
Some of the shorthands that will be declared by the language definition files have to be usable in both text and math mode. To achieve this the helper macro \textormath is provided.

\textormath
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.

\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 sure {} and \protect are taken into account in this new top level.

\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 sure {} and \protect are taken into account in this new top level.

\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 sure {} and \protect are taken into account in this new top level.

\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 sure {} and \protect are taken into account in this new top level.
\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].

\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 \aliasshorthand\{}\}/ is \active@prefix /\active@char/, so we still need to let the latter to \active@char". 

\def\aliasshorthand#1#2{\bbl@ifshorthand{#2}{}{\expandafter\ifx\csname active@char\string#2\endcsname\relax\if\document\@notprerr\@notshorthand{#2}\else\initiate@active@char{#2}\bbl@ccarg\let{active@char\string#2}{active@char\string#1}\bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}\bbl@activate{#2}\fi\fi}{\bbl@error{Cannot declare a shorthand turned off (\string#2)}{Sorry, but you cannot use shorthands which have been\% turned off in the package options}}

\@notshorthand \def\@notshorthand#1{\bbl@error{The character '\string #1' should be made a shorthand character;\% add the command \string\useshorthands\string{#1\string} to the preamble.\% I will ignore your instruction}\% {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 \@nil at the end to denote the end of the list of characters.

\shorthandoff \newcommand*\shorthandoff[1]{{\bbl@switch@sh\@ne#1\@nnil}}
\DeclareRobustCommand*\shorthandoff{\ifstar\bbl@switch@sh\tw@\else\bbl@switch@sh\z@\fi\bbl@switch@sh\@ne#1\@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 category code and the original definition, saved in @initiate@active@char, are restored.

```latex
\def\bbl@switch@sh#1#2{% 
  \ifx#2\@nnil\else 
    \bbl@ifunset{bbl@active@\string#2}{} 
  \fi 
  \ifcase\#1% off, on, off* 
    \catcode`#212\relax 
  \or 
    \catcode`#2\active 
    \bbl@ifunset{bbl@shdef@\string#2}{} 
    \bbl@withactive{\expandafter\let\expandafter}#2\csname bbl@shdef@\string#2\endcsname 
    \bbl@csarg\let{shdef@\string#2}\relax 
  \or 
    \bbl@activate{#2} 
  \else 
    \bbl@deactivate{#2} 
  \fi 
  \bbl@afterfi 
\bbl@switch@sh#1% 
}
```

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

```latex
\def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh} 
\def\bbl@putsh#1{% 
  \bbl@ifunset{bbl@active@\string#1}{} 
  \csname bbl@active@\string#1\endcsname 
  \def\putsh@i#1#2\@nnil{\csname\language@group @sh@\string#1@#2\endcsname} 
  \ifx\bbl@opt@shorthands\@nnil\else 
    \let\bbl@s@initiate@active@char\initiate@active@char 
    \def\initiate@active@char#1{% 
      \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{} 
    } 
    \let\bbl@s@switch@sh\bbl@switch@sh 
    \def\bbl@switch@sh#1#2{% 
      \bbl@ifshorthand{#1}{\bbl@s@switch@sh{#1}{#2}}{} 
    } 
    \let\bbl@s@activate\bbl@activate 
    \def\bbl@activate#1{% 
      \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{} 
    } 
    \let\bbl@s@deactivate\bbl@deactivate 
    \def\bbl@deactivate#1{% 
      \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{} 
    } 
  \fi 
}
```

You may want to test if a character is a shorthand. Note it does not test whether the shorthand is on
One of the internal macros that are involved in substituting \prime for each right quote in math mode is \bbl@prim@s. This checks if the next character is a right quote. When the right quote is active, the definition of this macro needs to be adapted to look also for an active right quote; the hat could be active, too.

\begin{verbatim}
def\bbl@prim@s{\prime\futurelet\@let@token\bbl@pr@m@s}
def\bbl@if@primes#1#2{\ifx#1\@let@token\expandafter\@firstoftwo\else\ifx#2\@let@token\bbl@afterelse\expandafter\@firstoftwo\else\bbl@afterfi\expandafter\@secondoftwo\fi\fi}
\begin{group}
catcode`\^=7 \catcode`*=active \lccode`\^=`
catcode`\"=12 \catcode`\"=active \lccode`\"=\'
lowercase{}
gdef\bbl@pr@m@s{\bbl@if@primes\pr@@@s}{\bbl@if@primes\pr@@@t\egroup}}}
\end{group}
\end{verbatim}

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).

\initiate@active@char{-}
\declare@shorthand{system}{-}{\leavevmode\nobreak\ }
\bbl@activate{-}

\begin{verbatim}
\expandafter\def\csname OT1dqpos\endcsname{127}
\expandafter\def\csname T1dqpos\endcsname{4}
\end{verbatim}

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

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.

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

\begin{verbatim}
1571 \ifx\bbl@known@attribs\undefined
1572 \in@false
1573 \else
1574 \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
1575 \fi
1576 \ifin@
1577 \bbl@warning{%
1578 You have more than once selected the attribute ‘##1’\%
1579 for language #1. Reported}%
1580 \else
1581 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.
1582 \bbl@exp{%
1583 \bbl@add@list\bbl@known@attribs{\bbl@tempc-##1}}%
1584 \edef\bbl@tempa{\bbl@tempc-##1}%
1585 \expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes%
1586 {\csname\bbl@tempc @attr@##1\endcsname}%
1587 \else}}}
1588 \@onlypreamble\languageattribute
\end{verbatim}

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

\begin{verbatim}
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}}
\end{verbatim}

\bbl@declare@tribute 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 \texttt{extras...} for the current language is extended, otherwise the attribute will not work as its code is removed from memory at \texttt{\begin{document}}.

\begin{verbatim}
1593 \def\bbl@declare@tribute#1#2#3{%
1594 \bbl@xin@{,\bbl@known@attribs{\bbl@tempc-#1}},%\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}
\end{verbatim}

\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 \texttt{\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.

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

\bbl@ifknown@trib An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the \TeX-code to be executed when the attribute is known and the \TeX-code to be executed otherwise.
We first assume the attribute is unknown. Then we loop over the list of known attributes, trying to find a match.

```latex
\def\bbl@ifknown@ttrib#1#2{\let\bbl@tempa@secondoftwo\bbl@loopx\bbl@tempb{#2}{\expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}\ifin@\let\bbl@tempa@firstoftwo\else\fi}\bbl@tempa}
\bbl@clear@ttribs
```

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

```latex
\def\bbl@clear@ttribs{\ifx\bbl@attributes\@undefined\else\bbl@loopx\bbl@tempa{\bbl@attributes}{\expandafter\bbl@clear@ttrib\bbl@tempa.}\let\bbl@attributes\@undefined\fi}
\def\bbl@clear@ttrib#1-#2.{\expandafter\let\csname#1@attr@#2\endcsname\@undefined}
\AtBeginDocument{\bbl@clear@ttribs}
```

4.7 Support for saving macro definitions

To save the meaning of control sequences using \texttt{\global@state}, 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 \texttt{\selectlanguage} and \texttt{\originalTeX}). Note undefined macros are not undefined any more when saved – they are \texttt{\relax}ed.

```latex
\begin{verbatim}
\texttt{\babel@savecnt}\quad \texttt{\babel@beginsave}
\texttt{\bbl@savecnt\quad \bbl@beginsave}
\texttt{\bbl@savestore}\quad \texttt{\bbl@beginsave}\texttt{variable}
```

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

```latex
\def\bbl@save#1{\def\bbl@tempa\{,#1,\}\expandafter\bbl@add\expandafter\bbl@tempa\expandafter{\expandafter,\bbl@savedextras,}\expandafter\in@\bbl@tempa\ifin@\bbl@add\bbl@savedextras\{,#1,\}\bbl@carg\let{babel@\number\bbl@savecnt}#1\relax\toks@\expandafter{\originalTeX\let#1=}\bbl@exp{\def\\originalTeX{\the\toks@<babel@\number\bbl@savecnt>\relax}}\bbl@exp{\advance\bbl@savecnt\@ne\originalTeX\hastobeexpandable,i.e. you shouldn't let it to \relax.}
```

40
Some languages need to have \frenchspacing in effect. Others don't want that. The command \bbl@frenchspacing 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 \bbl@provide. This new method should be ideally the default one.

Some languages need to have \frenchspacing in effect. Others don't want that. The command \bbl@frenchspacing 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 \bbl@provide. This new method should be ideally the default one.

\bbl@trace{Short tags}

\textbf{4.8 Short tags}

This macro is straightforward. After zapping spaces, we loop over the list and define the macros \textlangle tag\rangle and \text\langle tag\rangle. Definitions are first expanded so that they don't contain \csname but the actual macro.
4.9 Hyphens

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.

\bbl@allowhyphens

This macro makes hyphenation possible. Basically its definition is nothing more than \nobreak \hskip 0pt plus 0pt.

\bbl@nullhyphen

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.

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.

\textsuperscript{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.

\begin{verbatim}
\def\bbl@usehyphen#1{\leavevmode\ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
\nobreak\hskip\z@skip}
\def\bbl@@usehyphen#1{\leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
\end{verbatim}

The following macro inserts the hyphen char.

\begin{verbatim}
\def\bbl@hyphenchar{%
\ifnum\hyphenchar\font=\m@ne
\babelnullhyphen
\else
\char\hyphenchar\font
\fi}
\end{verbatim}

Finally, we define the hyphen “types”. Their names will not change, so you may use them in \df's. After a space, the \mbox in \bbl@hy@nobreak is redundant.

\begin{verbatim}
\def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
\def\bbl@hy@@soft{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
\def\bbl@hy@hard{\bbl@usehyphen{\bbl@hyphenchar}}
\def\bbl@hy@@hard{\bbl@@usehyphen{\bbl@hyphenchar}}
\def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
\def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
\def\bbl@hy@repeat{%
\bbl@usehyphen{%
\discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
\def\bbl@hy@@repeat{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
\def\bbl@hy@empty{\hskip\z@skip}
\def\bbl@hy@@empty{\discretionary{}{}{}}
\end{verbatim}

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

\begin{verbatim}
\def\bbl@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bbl@allowhyphens}
\end{verbatim}

\section*{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 \latex and \xetex. The code is organized here with pseudo-guards, so we start with the basic commands.

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

\begin{verbatim}
\def\bbl@trace{\Multiencoding strings}
\def\bbl@toglobal#1{\global\let#1#1}
\end{verbatim}

The following option is currently no-op. It was meant for the deprecated \SetCase.

\begin{verbatim}
\DeclareOption{nocase}{}
\DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
\DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
\end{verbatim}
**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.

\onlypreamble\StartBabelCommands
\def\StartBabelCommands{%
\begingroup
\@tempcnta="7F
\def\bbl@tempa{%
\ifnum\@tempcnta>"FF\else
  \catcode\@tempcnta=11
  \advance\@tempcnta\@one
  \expandafter\bbl@tempa
\fi%
\bbl@tempa
⟨⟨ Macros local to BabelCommands ⟩⟩
\def\bbl@provstring##1##2{%\providecommand##1{##2}%
\bbl@toglobal##1}%
\global\let\bbl@scafter\@empty
\let\StartBabelCommands\bbl@startcmds%
\ifx\BabelLanguages\relax
\let\BabelLanguages\CurrentOption
\fi
\begingroup
\let\bbl@screset\@nnil % local flag - disable 1st stopcommands
\StartBabelCommands}%
\def\bbl@startcmds{%
\ifx\bbl@screset\@nnil\else
  \bbl@usehooks{stopcommands}{}%
\fi
\endgroup
\begingroup
\@ifstar
  {\ifx\bbl@opt@strings\@nnil
  \let\bbl@opt@strings\BabelStringsDefault
  \fi
  \bbl@startcmds@i}%
\bbl@startcmds@i}
\def\bbl@startcmds@ii[#1][]{%
  \let\SetString\@gobbletwo
  \let\bbl@stringdef\@gobbletwo
  \let\AfterBabelCommands\@gobble
  \ifx\@empty#1%
    \def\bbl@sc@label{generic}%
    \def\bbl@encstring##1##2{%\providecommanddefault##1{##2}%
      \bbl@stringdef\csname\string##1\endcsname}%
  \else
    \edef\bbl@L{\zap@space#1 \@empty}%
    \edef\bbl@G{\zap@space#2 \@empty}%
    \bbl@startcmds@i}
\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 (i.e., 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 (i.e., 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.
\newcommand\bbl@startcmds@i[1][[@empty]{}% 
\let\SetString\@gobbletwo
\let\bbl@stringdef\@gobbletwo
\let\AfterBabelCommands\@gobble
\ifx\@empty#1%
  \def\bbl@opt@strings{label}{generic}%
  \def\bbl@opt@encoding{#1}{##2}%
  \bbl@stringdef\csname\string?\endcsname
There are two versions of \texttt{\bbl@scswitch}. The first version is used when ldfs are read, and it makes sure \texttt{\langle group\rangle\langle language\rangle} is reset, but only once (\texttt{\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 \texttt{\bbl@forlang} loops \texttt{\bbl@L} but its body is executed only if the value is in \texttt{\BabelLanguages} (inside babel) or \texttt{\date\langle language\rangle} is defined (after babel has been loaded). There are also two version of \texttt{\bbl@forlang}. The first one skips the current iteration if the language is not in \texttt{\BabelLanguages} (used in ldfs), and the second one skips undefined languages (after babel has been loaded).

\begin{verbatim}
\def\bbl@forlang#1#2{\bbl@for#1\bbl@L{%\bbl@xin{,#1,}{,\BabelLanguages,}#2}}
\def\bbl@scswitch{\bbl@screset}{\bbl@setstring}{\bbl@stringdef}{\bbl@encstring}{\ifx\@empty\empty}{\error{Missing group for string \string##1}{You must assign strings to some category, typically\% captions or extras, but you set none}}}
\end{verbatim}
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 (i.e., 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.

\def\bbl@setstring#1#2{%
  \bbl@forlang\bbl@tempa{%\edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
    \bbl@ifunset{\bbl@LC}{\bbl@exp{\global\bbl@add\bbl@G\bbl@tempa\bbl@LC}}{}
  }
  \def\BabelString{#2}
  \bbl@usehooks{stringprocess}{}
  \expandafter\bbl@stringdef\csname\bbl@LC\endcsname{\BabelString}
}

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

\def\bbl@stringloop{\count@\z@
obracket Insert \the\count@ remaining\
bbl@loop\bbl@tempa{% empty items and spaces are ok
  \advance\count@\@ne
  \toks@\exp{%
    \\\SetString\bbl@templ\romannumeral\count@\{\the\toks@\%
    \count@=\the\count@\relax
}}}

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.

Delaying code

Now the definition of \AfterBabelCommands when it is activated.

\def\bbl@aftercmds#1{\toks@\exp{%
  \\\AfterBabelCommands\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.

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.

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

The following package options control the behavior of hyphenation mapping.

The command \SetCase is deprecated. Currently it consists in a definition with a hack just for backward compatibility in the macro mapping.
Initial setup to provide a default behavior if hyphenmap is not set.

```latex
\AtEndOfPackage{% 
  \ifx\bbl@opt@hyphenmap\undefined 
    \bbl@xin{},\bbl@language@opts% 
  \ch@rdef\bbl@opt@hyphenmap\ifin4\else\ne\fi 
  \fi}
```

This section 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.

```latex
\newcommand\setlocalecaption{% TODO. Catch typos. 
  \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
\def\bbl@setcaption@x#1#2#3{% language caption-name string 
  \bbl@trim@def\bbl@tempa{#2}%.template{\bbl@tempa}% 
  \ifin@ \bbl@ini@captions@template{#3}{#1}% 
  \else \edef\bbl@tempd{\expandafter\expandafter\expandafter 
    \strip@prefix\expandafter\meaning\csname captions#1\endcsname} 
    \bbl@xin@{\expandafter\string\csname #2name\endcsname} 
    \bbl@tempd% Renew caption 
  \bbl@xign{\string\bbl@scset}{\bbl@tempd}% 
  \ifin@ \bbl@exp{\bbl@ifsamestring{\bbl@tempa}{\languagename} 
    {\bbl@scset#2name\bbl@tempd}% New 
  \else \bbl@xign{\string\bbl@scset}{\bbl@tempd}% New way 
  \bbl@xign@{\string\bbl@scset}{\bbl@tempd}% New way 
  \bbl@exp{% 
    \bbl@add\captions#1{\bbl@scset\bbl@tempd}{\languagename} 
    \bbl@scset\bbl@tempd}{\languagename} 
  \ifin@ \bbl@add\captions#1{\bbl@scset\bbl@tempd}{\languagename} 
  \bbl@xign@{\string\bbl@scset}{\bbl@tempd}% New way 
  \bbl@exp{% 
    \bbl@add\captions#1{\bbl@scset\bbl@tempd}{\languagename} 
    \bbl@scset\bbl@tempd}{\languagename} 
  \ifin@ \bbl@add\captions#1{\bbl@scset\bbl@tempd}{\languagename} 
  \bbl@xign@{\string\bbl@scset}{\bbl@tempd}% New way 
  \bbl@exp{% 
    \bbl@add\captions#1{\bbl@scset\bbl@tempd}{\languagename} 
    \bbl@scset\bbl@tempd}{\languagename} 
  \ifin@
```
4.11 Macros common to a number of languages

\texttt{\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.

\texttt{\bbl@trace{Macros related to glyphs}}
\texttt{\def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}\}%
\dimen\z@\ht\z@\advance\dimen\z@ -\ht\tw@$%
\setbox\z@\hbox{\lower\dimen\z@ \box\z@}\ht\z@\ht\tw@ \dp\z@\dp\tw@}}
\texttt{\save@sf@q}
The macro \texttt{\save@sf@q} is used to save and reset the current space factor.

\texttt{\def\save@sf@q#1{\leavevmode
\begingroup
\edef\@SF{\spacefactor\the\spacefactor}#1\@SF
\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 \texttt{T1enc.def}.

4.12.1 Quotation marks

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

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

\texttt{\ProvideTextCommandDefault{\quotedblbase}{%
\UseTextSymbol{OT1}{\quotedblbase}}}

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

\texttt{\ProvideTextCommand{\quotesinglbase}{OT1}{%
\save@sf@q{\set@low@box{\textquoting\right}\}%
\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.

\texttt{\ProvideTextCommandDefault{\quotesinglbase}{%
\UseTextSymbol{OT1}{\quotesinglbase}}}

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

\texttt{\ProvideTextCommand{\guillemetleft}{OT1}{%
\ifmmode
\ll
\else
\save@sf@q{\nobreak
\raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
\fi}
\ProvideTextCommand{\guillemetright}{OT1}{%
\ifmmode
\gg
\else
\save@sf@q{\nobreak
\raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
\fi}
\ProvideTextCommand{\guillemotleft}{OT1}{%
\ProvideTextCommand{\guillemoretright}{OT1}{%
Makes sure that when an encoding other than OT1 or T1 is used these glyphs can still be typeset.

\providecommand\guillemotleft{OT1}{\guillemetleft}
\providecommand\guillemotright{OT1}{\guillemetright}
\providecommand\guilsinglleft{OT1}{\guilsinglleft}
\providecommand\guilsinglright{OT1}{\guilsinglright}

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

4.12.2 Letters

\ij \IJ

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

\ij\IJ

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

\ij\IJ

50
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).

\begin{verbatim}
def\crrtic@:\hrule height0.1ex width0.3em
def\ddj@{%\setbox0\hbox{d}\dimen@=\ht0\advance\dimen@1ex\dimen@.45\dimen@\dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@\advance\dimen@ii.5ex\leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
def\DDJ@{%\setbox0\hbox{D}\dimen@=.55\ht0\dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@\advance\dimen@ii.15ex % correction for the dash position\advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font\dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@\leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
\def\dj{\ddj@ d}\def\DJ{\DDJ@ D}
\end{verbatim}

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

\begin{verbatim}
\ProvideTextCommandDefault{\dj}{%\UseTextSymbol{OT1}{\dj}}\ProvideTextCommandDefault{\DJ}{%\UseTextSymbol{OT1}{\DJ}}
\end{verbatim}

\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.

\begin{verbatim}
\DeclareTextCommand{\SS}{OT1}{SS}\ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
\end{verbatim}

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 math mode. 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.
\glqq The ‘german’ double quotes.
\begin{verbatim}
\ProvideTextCommandDefault{\glq}{%\textormath{\textquotesinglebase}{\mbox{\textquotesinglebase}}}\ProvideTextCommandDefault{\glqq}{%\textormath{\textdoublequote}{\mbox{\textdoublequote}}
\end{verbatim}

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

\begin{verbatim}
\ProvideTextCommand{\grq}{T1}{%\textormath{\kern.07em\relax}}\Save@sf@q{%\kern-.0125em\textormath{\textdoublequote}{\mbox{\textdoublequote}}\kern.07em\relax}\ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}{\grq}}\end{verbatim}

\begin{verbatim}
\ProvideTextCommandDefault{\glq}{%\textormath{\textquotesinglebase}{\mbox{\textquotesinglebase}}}\ProvideTextCommandDefault{\glqq}{%\textormath{\textdoublequote}{\mbox{\textdoublequote}}}
\end{verbatim}

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

\begin{verbatim}
\ProvideTextCommand{\grqq}{T1}{%\textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}\ProvideTextCommand{\grqq}{TU}{%\textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}\ProvideTextCommand{\grqq}{OT1}{%\textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}\Save@sf@q{%\kern-.0125em\textormath{\textdoublequote}{\mbox{\textdoublequote}}\kern.07em\relax}\ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}{\grqq}}\end{verbatim}
The 'french' single guillemets.

The 'french' double guillemets.

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.

\umlaut{high}
\umlaut{low}

To be able to provide both positions of " we provide two commands to switch the positioning, the default will be \umlaut{high} (the normal positioning).

\def\umlaut{high}{\def\bbl@umlauta##1{\leavevmode\bgroup\accent\csname\f@encoding dqpos\endcsname##1\bbl@allowhyphens\egroup}}
\def\umlaut{low}{\def\bbl@umlauta{\protect\lower@umlaut}}
\umlaut{high}
\umlaut{low}

\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 \texttt{dimen} register.

\expandafter\if%\csname U@D\endcsname\relax
\csname newdimen\endcsname\U@D
\fi

The following code fools TreX'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 \texttt{accent} primitive, reset the old x-height and insert the base character in the argument.
For all vowels we declare \" to be a composite command which uses \@umlauta or \@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 \@umlauta and/or \@umlaute for a language in the corresponding ldf (using the babel switching mechanism, of course).

\AtBeginDocument{%
\DeclareTextCompositeCommand{"}{OT1}{a}{\@umlauta{a}}%
\DeclareTextCompositeCommand{"}{OT1}{e}{\@umlaute{e}}%
\DeclareTextCompositeCommand{"}{OT1}{i}{\@umlaute{i}}%
\DeclareTextCompositeCommand{"}{OT1}{\i}{\@umlaute{\i}}%
\DeclareTextCompositeCommand{"}{OT1}{o}{\@umlauta{o}}%
\DeclareTextCompositeCommand{"}{OT1}{u}{\@umlauta{u}}%
\DeclareTextCompositeCommand{"}{OT1}{A}{\@umlauta{A}}%
\DeclareTextCompositeCommand{"}{OT1}{E}{\@umlaute{E}}%
\DeclareTextCompositeCommand{"}{OT1}{I}{\@umlaute{I}}%
\DeclareTextCompositeCommand{"}{OT1}{O}{\@umlauta{O}}%
\DeclareTextCompositeCommand{"}{OT1}{U}{\@umlauta{U}}%
}

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

\ifx\l@english\@undefined
\chardef\l@english\z@
\fi
\% The following is used to cancel rules in ini files (see Amharic).
\ifx\l@unhyphenated\@undefined
\newlanguage\l@unhyphenated
\fi

4.13 Layout

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

4.15 Creating and modifying languages

Continue with \LaTeX only.
\texttt{\textbackslash 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 \texttt{.ldf} files.

54
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.

```latex
\atthispointallparametersaredefinedif'import'.Nowweweexecute\somecodel dependingonthem.

\ifx\bbl@inidata@\emptyelse
\bbl@load@basic{#2}\fi
\ifx\bbl@KVP@captions@\emptyelse
\bbl@load@basic{#2}\fi
\if\bbl@KVP@script@\emptyelse
\bbl@csarg\edef{sname@#2}{\bbl@KVP@script}\fi
\if\bbl@KVP@language@\emptyelse
\bbl@csarg\edef{lname@#2}{\bbl@KVP@language}\fi
\if\bbl@engine@\or
\bbl@ifunset{bbl@chrng@\languagename}{\directlua{
Babel.set_chranges_b('\bbl@cl{sbcp}', 'bbl@cl{chrng}')}}\fi
\if\bbl@KVP@onchar@\emptyelse
\bbl@load@basic{#2}\fi
\if\bbl@KVP@language@\emptyelse
\bbl@load@basic{#2}\fi
\if\bbl@KVP@onchar@\emptyelse
\bbl@load@basic{#2}\fi
\if\bbl@KVP@onchar@\emptyelse
\bbl@load@basic{#2}\fi
\if\bbl@KVP@onchar@\emptyelse
\bbl@load@basic{#2}\fi
\if\bbl@KVP@onchar@\emptyelse
\bbl@load@basic{#2}\fi
\if\bbl@KVP@onchar@\emptyelse
\bbl@load@basic{#2}\fi
\if\bbl@KVP@onchar@\emptyelse
\bbl@load@basic{#2}\fi
\if\bbl@KVP@onchar@\emptyelse
\bbl@load@basic{#2}\fi
```

56
\bbl@xin{ letters }{ \bbl@KVP@onchar\space} %
\ifin@ \directlua{
  Babel.locale_props[\the\localeid].letters = true
}\fi
\bbl@xin{ ids }{ \bbl@KVP@onchar\space} %
\ifin@ \directlua{
  \if\bbl@starthyphens\@undefined % Needed if no explicit selection
  \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens} %
  \bbl@exp{\bbl@add\bbl@starthyphens
    {\bbl@patterns@lua{\languagename}}} %
  \if\the\localeid\space % TODO - error/warning if no script
    \begin{verbatim}
    if Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@languagename}\space
    \end{verbatim}
  \fi
\fi
\bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languagename}}} %
\if\bbl@mapselect\@undefined % TODO. almost the same as mapfont
  \AtBeginDocument{%
    \bbl@patchfont{%bbl@mapselect}{\selectfont}%
    \def\bbl@mapselect{%
      \let\bbl@switchfont
      \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
        \directlua{
          Babel.locale_props[\the\csname bbl@id@@##1\endcsname]...
        }%}
      \fi
    }%}
  \bbl@exp{\bbl@add\bbl@mapselect{%bbl@mapdir{\languagename}}} %
\fi
\if\bbl@mapselect\@undefined % TODO. See onchar.
  \AtBeginDocument{%
    \bbl@patchfont{%bbl@mapselect}{\selectfont}%
    \def\bbl@mapselect{%
      \let\bbl@switchfont
      \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
        \directlua{
          Babel.locale_props[\the\csname bbl@id@@##1\endcsname]...
        }%}
      \fi
    }%}
\fi
\if\bbl@mapselect\@undefined % TODO. almost the same as mapfont
  \AtBeginDocument{%
    \bbl@patchfont{%bbl@mapselect}{\selectfont}%
    \def\bbl@mapselect{%
      \let\bbl@switchfont
      \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
        \directlua{
          Babel.locale_props[\the\csname bbl@id@@##1\endcsname]...
        }%}
      \fi
    }%}
\fi
\if\bbl@mapselect\@undefined % TODO. almost the same as mapfont
  \AtBeginDocument{%
    \bbl@patchfont{%bbl@mapselect}{\selectfont}%
    \def\bbl@mapselect{%
      \let\bbl@switchfont
      \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
        \directlua{
          Babel.locale_props[\the\csname bbl@id@@##1\endcsname]...
        }%}
      \fi
    }%}
\fi
% == mapfont ==
% For bidi texts, to switch the font based on direction
\if\bbl@KVP@mapfont\@nil\else
  \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{%}
  \bbl@error{Option 'bbl@KVP@mapfont' unknown for} %
  \bbl@error{mapfont. Use 'direction'.} %
  \bbl@error{See the manual for details.} %
\fi
\bbl@exp{\bbl@add\bbl@mapselect{%bbl@mapdir{\languagename}}} %
\fi
% TODO - catch non-valid values
% % = mapfont =
\bbl@patchfont{\{\bbl@mapselect\}}\%
\def\bbl@mapselect{%
  \let\bbl@mapselect\relax
  \edef\bbl@prefontid{\fontid\font}
}\def\bbl@mapdir##1{%
  {\def\languagename{##1}%
    \let\bbl@ifrestoring\@firstoftwo % avoid font warning
    \bbl@switchfont
    \directlua{Babel.fontmap
      [[\the\csname bbl@wdir@##1\endcsname]]
      [[\bbl@prefontid]\fontid\font]}}%
}\fi
\bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languagename}}}
\fi
% == Line breaking: intraspace, intrapenalty ==
% For CJK, East Asian, Southeast Asian, if interspace in ini
% We can override the ini or set
\ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
  \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
\fi
\bbl@provide@intraspace
% == Line breaking: CJK quotes == TODO -> @extras
\ifcase\bbl@engine\or
  \bbl@xin{/c}{/\bbl@cl{lnbrk}}%
  \ifin@
    \bbl@ifunset{bbl@quote@\languagename}{}%
    \directlua{
      Babel.locale_props[\the\localeid].cjk_quotes = {}
      for c in string.utfvalues(%
        [[\csname bbl@quote@\languagename\endcsname]]) do
        if Babel.cjk_characters[c].c == 'qu' then
          Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
        end
        cs = ( cs == 'op') and 'cl' or 'op'
      end
    }%
  \fi
\fi
\bbl@provide@intraspace
% == Line breaking: justification ==
\ifx\bbl@KVP@justification\@nnil\else
  \let\bbl@KVP@linebreaking\bbl@KVP@justification
\fi
\fi
\ifx\bbl@KVP@linebreaking\@nnil\else
  \bbl@xin{},\bbl@KVP@linebreaking,\iend
  \ifin@
    \bbl@csarg\xdef
    {lnbrk@\languagename}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
  \fi
\fi
\bbl@xin{/e}{/\bbl@cl{lnbrk}}%
\ifin@
  \bbl@xin{}/k{/\bbl@cl{lnbrk}}
\fi
\bbl@arabicjust
\bbl@xin{/p}{/\bbl@cl{lnbrk}}%
\ifin@
  \AtBeginDocument{\@nameuse{bbl@tibetanjust}}
\fi
% == Line breaking: hyphenate.other.(locale|script) ==
\ifx\bbl@lbkflag\@empty
  \bbl@ifunset{bbl@hyotl@\languagename}{}%
  \bbl@csarg\bbl@replace{hyotl@\languagename}{,}{%\bbl@startcommands*{\languagename}{}%
  \bbl@csarg\bbl@foreach{hyotl@\languagename}{%\ifcase\bbl@engine%
Depending on whether or not the language exists (based on `\date<language>`), we define two macros. Remember `\bbl@startcommands` opens a group.

\def\bbl@provide@new#1{\bbl@startcommands...}
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.)
The hyphenrules option is handled with an auxiliary macro. This macro is called in three cases:
when a language is first declared with \bbl\provide, with hyphenrules and with import.
\def\bbl\provide\hyphens#1{% 
\@tempcnta\m@ne % a flag 
\ifdef\bbl\KVP\hyphenrules\@nil\else 
\bbl\foreach\bbl\KVP\hyphenrules{% 
\ifnum\@tempcnta\m@ne % if not yet found 
\bbl\ifsamestring{#1}{% 
{\bbl\carg\addlanguage{1}}% 
} 
\bbl\ifunset{1} % After a possible + 
{} 
{\@tempcnta\nameuse{1}}% 
\fi} 
\fi} 
\ifnum\@tempcnta\m@ne 
\bbl\warning{% 
Requested 'hyphenrules' for '{1}' not found:\% 
\bbl\KVP\hyphenrules.\% 
Using the default value. Reported}% 
\fi 
\fi 
\ifnum\@tempcnta\m@ne % if no opt or no language in opt found 
\ifdef\bbl\KVP\captions\@nil % TODO. Hackish. See above. 
\bbl\ifunset{\bbl\hyphr}{} % use value in ini, if exists 
{\bbl\exp{\bbl\ifblank{\bbl\cs{hyphr}}}{} 
{\@tempcnta\nameuse{1}}% 
\fi} 
\fi 
\bbl\ifunset{1} % if hyphenrules found: 
{\@tempcnta\nameuse{1}{\bbl\cl{hyphr}}} 
\fi 
\bbl\ifunset{1} 
{\ifdef\@tempcnta\m@ne 
\bbl\carg\adddialect{1}\language 
\else 
\bbl\carg\adddialect{1}\@tempcnta 
\fi} 
{\ifdef\@tempcnta\m@ne\else 
{\global\bbl\carg\chardef{1}\@tempcnta 
\fi} 
The reader of babel-...\text files. We reset temporarily some catcodes.
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@readini.

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 \\bbl@provide it’s either 1 or 2.
{There is no ini file for the requested language\%}
(#1: \languagename). Perhaps you misspelled it or your\%
installation is not complete.)\%
{Fix the name or reinstall babel.}\%
\else
% == Store ini data in \bbl@inidata ==
\catcode`\[=12 \catcode`\]=12 \catcode`\&=12
\catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
\bbl@info{Importing
\ifcase#2font and identification \or basic \fi
data for \languagename\%
from babel-#1.ini. Reported}\%
\ifnum#2=\z@
\global\let\bbl@inidata\@empty
\let\bbl@inistore\bbl@inistore@min % Remember it's local
\fi
\def\bbl@section{identification}\
\bbl@exp{\bbl@inistore tag.ini=#1\@@}
\bbl@inistore load.level=#2\@@
\bbl@loop@ini
% == Process stored data ==
\bbl@csarg\xdef{lini@\languagename}{#1}\
\bbl@read@ini@aux
% == 'Export' data ==
\bbl@ini@exports{#2}\
\global\bbl@csarg\let{inidata@\languagename}\bbl@inidata\
\global\let\bbl@inidata\@empty
\bbl@exp{\bbl@add@list\bbl@ini@loaded{\languagename}}\
\bbl@toglobal\bbl@ini@loaded
\fi
\closein\bbl@readstream\
\def\bbl@read@ini@aux{%
\let\bbl@savestrings\@empty
\let\bbl@savetoday\@empty
\let\bbl@savedate\@empty
\def\bbl@elt##1##2##3{%\def\bbl@section{##1}\
\in@{=date.}{=##1}% Find a better place
\ifin@
\bbl@ifunset{bbl@inikv@##1}\
{\bbl@ini@calendar{##1}}\
\bbl@ifunset{bbl@inikv@##1}{\csname bbl@inikv@##1\endcsname{##2}{##3}}}\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.
\def\bbl@extend@ini@aux#1{%\bbl@startcommands*{#1}{captions}\
% Activate captions/... and modify exports
\bbl@csarg\def{inikv@captions.licr}{#1}{#2}{#3}\
\setlocalecaption{#1}{#2}{#3}\
\def\bbl@inikv@captions##1##2{\bbl@ini@captions@aux{##1}{##2}}\
\def\bbl@stringdef##1##2{%\bbl@ifunset{bbl@@kv@##2}{}\
{\expandafter\ifx\csname bbl@@kv@##2\endcsname\@empty\else
{\bbl@exp{\bbl@add@list}\bbl@ini@loaded{\languagename}}\
\bbl@toglobal\bbl@ini@loaded\fi}}%}
% As with \bbl@read@ini, but with some changes

64
\bbl@readini@aux
\bbl@ini@exports\tw@
% Update inidata@lang by pretending the ini is read.
\def\bbl@elt##1##2##3{%  
\def\bbl@section{##1}%  
\bbl@inilinear##2=##3\bbl@inilinear}%
csname bbl@inidata@#1\endcsname
\global\bbl@csarg\let{inidata@#1}\bbl@inidata
StartBabelCommands*{#1}{date}% And from the import stuff
\def\bbl@stringdef##1##2{\gdef##1{##2}}%
bbl@savetoday
bbl@savedate
\bbl@endcommands}

As somewhat hackish tool to handle calendar sections. TODO. To be improved.
\def\bbl@ini@calendar#1{%  
\lowercase{\def\bbl@tempa{=#1=}}%  
bbl@replace\bbl@tempa{=date.gregorian}{}%  
bbl@replace\bbl@tempa{=date.}{}%  
\in@{.licr=}{#1=}%  
\ifin@  
\ifcase\bbl@engine  
bbl@replace\bbl@tempa{.licr=}{}%  
\else  
\let\bbl@tempa\relax  
\fi  
\fi  
\ifx\bbl@tempa\relax\else  
bbl@replace\bbl@tempa{=}{}%  
\ifx\bbl@tempa\@empty\else  
xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%  
\fi  
\fi}
\bbl@renewinikey#1/#2\@@#3{%  
\edef\bbl@tempa{\zap@space #1 \@empty}% section  
\edef\bbl@tempb{\zap@space #2 \@empty}% key  
\bbl@trim\toks@{#3}% value  
\bbl@exp{%  
\edef\bbl@key@list{\bbl@key@list \bbl@tempa/%\bbl@tempb;}%  
\g@addto@macro{\bbl@inidata}{%  
\\\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.
\def\bbl@exportkey#1#2#3{%  
\bbl@ifunset{bbl@@kv@#2}%  
{\bbl@csarg\gdef{#1@\languagename}{#3}}%  
{\expandafter\ifx\csname bbl@@kv@#2\endcsname\@empty  
\bbl@exp{%  
\\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%  
\\g@addto@macro{\bbl@inidata}{%  
\\\bbl@elt{\bbl@tempa/\bbl@tempb}{\the\toks@}}}%}

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

\def\bbl@iniwarning#1{%  
  \bbl@ifunset{bbl@@kv@identification.warning#1}{}%  
  {\bbl@warning{  
  From babel-\bbl@cs{lini@languagename}.ini:\%  
  \bbl@cs[@kv@identification.warning#1]\%  
  Reported }}}%
A shared handler for key=val lines to be stored in \bbl@kv@<section>.<key>.

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

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.

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'.

\def\bbl@maybextx{-\bbl@csarg\ifx{extx@\languagename}\@empty x-\fi}
\def\bbl@inikv@counters#1#2{%
\bbl@ifsamestring{#1}{digits}%
\bbl@error{The counter name 'digits' is reserved for mapping\%
\bbl@exp{\g@addto@macro\bib@release@casing{\bbl@casemapping{\unexpanded{#2}}}}}%
\in@{.1$}{#1$}\
\ifin@
\bbl@replace{.1}{}\% eg, .1u = .1\%\bbl@csarg\protected@xdef{cntr@#1@\languagename}{\bbl@tempb*}\%\else\toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
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.

The auxiliary macro for captions define \caption{name}.

Labels. Captions must contain just strings, no format at all, so there is new group in ini files.
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.
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
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 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).
range 0-9999.}
{(There is little you can do. Sorry.)%\fi\fi\fi\fi}}
\newcommand\BabelDateyyyy[1]{{\number#1}} % TODO - add leading 0
\newcommand\BabelDateU[1]{{\number#1}}%
def\bbl@replace@finish@iii#1{%
bbl@exp{\def\#1####1####2####3{\the\toks@}}}
def\bbl@TG@@date{% \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}% \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}% \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}% \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}% \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}% \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}% \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}% \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}% \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}% \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}% \bbl@replace\bbl@toreplace{[U]}{\BabelDateU{####1}}% \bbl@replace\bbl@toreplace{[y|}{\bbl@datecntr[####1|}% \bbl@replace\bbl@toreplace{[U|}{\bbl@datecntr[####1|}% \bbl@replace\bbl@toreplace{[m|}{\bbl@datecntr[####2|}% \bbl@replace\bbl@toreplace{[d|}{\bbl@datecntr[####3|}% \bbl@replace@finish@iii\bbl@toreplace}
\def\bbl@datecntr{\expandafter\bbl@xdatecntr\expandafter}
\def\bbl@xdatecntr[#1|#2]{\localenumeral{#2}{#1}}

Transforms.
\bbl@carg\let{inikv@transforms.prehyphenation}\bbl@inikv
\bbl@carg\let{inikv@transforms.posthyphenation}\bbl@inikv
\def\bbl@transforms@aux#1#2#3,#4,#5\relax{#1[#2]{#3}{#4}{#5}}
\begingroup % A hack. TODO. Don’t require an specific order
\catcode`%=12
\catcode`&=14
\gdef\bbl@transforms#1#2#3{&%
directlua{local str = \#2==\}
str = str:gsub('%.%d+%.%d+$', '')
token.set_macro('babeltempa', str)
}&%
\def\bbl@babetempc()&%
\bbl@xin@{,\bbl@KVP@transforms,}&%
\ifin@&text{\bbl@KVP@transforms,}&%
\fi\bbl@foreach\bbl@KVP@transforms{&%
\bbl@xin@{:\babeltempa,}{,##1,}&%
\ifin@ &% font:font:transform syntax
\directlua{local t ={}
for m in string.gmatch('##1..', '(.-):') do
  table.insert(t, m)
end
table.remove(t)
token.set_macro('babeltempc', 'fonts=' .. table.concat(t, ' '))
}&%
\fi}&%
\fi\bbl@foreach\bbl@KVP@transforms{&%
\bbl@xin@{\bbl@KVP@transforms,}[#1],&%
\ifin@ &% font:font:transform syntax
\directlua{%
  local t ={}
  for m in string.match('[['..', ']', '([{], ') do
    table.insert(t, m)
  end
  table.remove(t)
  token.set_macro('babeltempc', ',fonts=' .. table.concat(t, ' '))
}&%
\fi}&%
\in@(.0$){#2$}&%
\ifin@ &% (attribute) syntax
\directlua{%
  local str = string.match('[[[\bbl@KVP@transforms]]],

72
Language and Script values to be used when defining a font or setting the direction are set with the following macros.

\def\bbl@provide@lsys#1{\bbl@ifunset{bbl@lname@#1}{}\bbl@csarg\let{lsys@#1}\@empty\bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}\bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}\bbl@ifunset{bbl@lname@#1}{}\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}\ifcase\bbl@engine\or\or\bbl@ifunset{bbl@prehc@#1}{}\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}{}\ifx\bbl@xenohyph\@undefined\global\let\bbl@xenohyph\bbl@xenohyph@d\ifx\AtBeginDocument\@notprerr\expandafter\@secondoftwo % to execute right now\AtBeginDocument{\bbl@patchfont{\bbl@xenohyph}\select@language{\languagename}}\fi\fi\bbl@csarg\bbl@toglobal{lsys@#1}}

\def\bbl@xenohyph@d{\bbl@ifset{bbl@prehc@\languagename}{\ifnum\hyphenchar\font=\defaulthyphenchar\iffontchar\font\bbl@cl{prehc}\relax\hyphenchar\font\bbl@cl{prehc}\relax\else\iffontchar\font^200B\hyphenchar\font^200B\else\bbl@warning{Neither 0 nor ZERO WIDTH SPACE are available\% in the current font, and therefore the hyphen\% will be printed. Try changing the fontspec's\% 'HyphenChar' to another value, but be aware\% this setting is not safe (see the manual).\% Reported}\hyphenchar\font\defaulthyphenchar}}\fi}
The following ini reader ignores everything but the identification section. It is called when a font is defined (i.e., 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).

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

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

```latex
\newcommand{\localenumeral}[2]{\bbl@cs{cntr@#1@\languagename}{#2}}
\def{\bbl@localecntr#1@2}{localenumeral{#2}{#1}}
\newcommand{\localecounter}[2]{\bbl@localecntr{\number\csname c@#2\endcsname}{#1}}
\def{\bbl@alphnumeral#1@#2}{\expandafter{\bbl@alphnumeral@i\number#2\@@{#1}}}
\def{\bbl@alphnumeral@i#1#2#3#4#5#6#7#8}{\ifcase\@car#8\@nil\or \bbl@alphnumeral@ii{#9}000000#1\or \bbl@alphnumeral@ii{#9}00000#1#2\or \bbl@alphnumeral@ii{#9}0000#1#2#3\else \bbl@alphnumeral@invalid{>9999}\fi}
\def{\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8}{\bbl@ifunset{bbl@cntr@#1.F.@\number#5#6#7#8\@\languagename}{}{\bbl@cs{cntr@#1.4@\languagename}{#5}\bbl@cs{cntr@#1.3@\languagename}{#6}\bbl@cs{cntr@#1.2@\languagename}{#7}\bbl@cs{cntr@#1.1@\languagename}{#8}\ifnum#6#7#8<\z@ \bbl@ifunset{bbl@cntr@#1.S.321@\languagename}{}{}\bbl@cs{cntr@#1.F.@\number#5#6#7#8\@\languagename}}}
\def{\bbl@alphnumeral@invalid#1}{\bbl@error{Alphabetic numeral too large (#1)}{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.

```latex
\newcommand{\localeinfo}[1]{\ifx*#1\@empty % TODO. A bit hackish to make it expandable.
\bbl@afterelse{\localeinfo{}%}
\else
\localeinfo{\bbl@error{I've found no info for the current locale.\% The corresponding ini file has not been loaded\% Perhaps it doesn't exist\% (See the manual for details.)}%}
{#1}%
% \@namedef{bbl@info@name.locale}{lcname}
% \@namedef{bbl@info@name.english}{elname}
% \@namedef{bbl@info@name.opentype}{lname}
% \@namedef{bbl@info@tag.bcp47}{tbcp}
% \@namedef{bbl@info@script.name}{esname}
% \@namedef{bbl@info@script.name.opentype}{sname}
% \@namedef{bbl@info@script.tag.bcp47}{sname}
% \@namedef{bbl@info@script.tag.opentype}{sname}
% \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
% \@namedef{bbl@info@variant.tag.opentype}{vbcp}
```
\LaTeX needs to know the BCP 47 codes for some features. For that, it expects $\text{BCPdata}$ to be defined. While language, region, script, and variant are recognized, extension fields for singletons may change.

\begin{verbatim}
\ifcase\bbl@engine % Converts utf8 to its code (expandable)
  \def\bbl@utftocode#1{\the\numexpr\decode@UTFviii#1\relax}
\else
  \def\bbl@utftocode#1{\expandafter`\string#1}
\fi
\fi\renewcommand\BCPdata{}
\ifx\renewcommand\@undefined\else % For plain. TODO. It's a quick fix
  \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty}
\def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
  \@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
  {\bbl@bcpdata@ii{#6}\bbl@main@language}%
  {\bbl@bcpdata@ii{#1#2#3#4#5#6}\languagename}}%
\def\bbl@bcpdata@ii#1#2{%
  \bbl@ifunset{bbl@info@#1.tag.bcp47}%
  {\bbl@error{Unknown field '#1' in $\string\BCPdata$.\% Perhaps you misspelled it.}%
    \{See the manual for details.\}}%
  {\bbl@ifunset{bbl@cs{\csname bbl@info@#1.tag.bcp47\endcsname}}{}%
    \bbl@cs{\csname bbl@info@#1.tag.bcp47\endcsname}}%
\fi
\n\newcommand\BabelUppercaseMapping[3]{% 1:variant
  \DeclareUppercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
\newcommand\BabelTitlecaseMapping[3]{% 1:variant
  \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
\newcommand\BabelLowercaseMapping[3]{% 1:variant
  \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
\fi
\n\newcommand\BabelUppercaseMapping[3]{% 1:variant
  \DeclareUppercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
\newcommand\BabelTitlecaseMapping[3]{% 1:variant
  \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
\newcommand\BabelLowercaseMapping[3]{% 1:variant
  \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
\fi
\n\newcommand\BabelUppercaseMapping[3]{% 1:variant
  \DeclareUppercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
\newcommand\BabelTitlecaseMapping[3]{% 1:variant
  \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
\newcommand\BabelLowercaseMapping[3]{% 1:variant
  \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
\fi
\n\newcommand\BabelUppercaseMapping[3]{% 1:variant
  \DeclareUppercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
\newcommand\BabelTitlecaseMapping[3]{% 1:variant
  \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
\newcommand\BabelLowercaseMapping[3]{% 1:variant
  \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
\fi
\end{verbatim}
With version 3.75 \BabelEnsureInfo is executed always, but there is an option to disable it.

\begin{itemize}
\item More package options
\end{itemize}

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.

\begin{itemize}
\item LocaleForEach
\end{itemize}

\begin{verbatim}
5 Adjusting the Babel behavior
A generic high level interface is provided to adjust some global and general settings.
\end{verbatim}
\def\bbl@adjust@lua#1#2{%
  \ifnum\currentgrouplevel=\z@%
    \directlua{ Babel.#2 }%
    \expandafter\expandafter\expandafter\@gobble%
  \fi%
}{%}
\def\bbl@error % The error is gobbled if everything went ok.
{Currently, \#1 related features can be adjusted only\%
  in the main vertical list.}%
{Maybe things change in the future, but this is what it is.}}
\@namedef{bbl@ADJ@bidi.mirroring@on}{%}
\@namedef{bbl@ADJ@bidi.mirroring@off}{%}
\@namedef{bbl@ADJ@bidi.text@on}{%}
\@namedef{bbl@ADJ@bidi.text@off}{%}
\@namedef{bbl@ADJ@bidi.math@on}{%}
\@namedef{bbl@ADJ@bidi.math@off}{%}
\@namedef{bbl@ADJ@linebreak.sea@on}{%}
\@namedef{bbl@ADJ@linebreak.sea@off}{%}
\@namedef{bbl@ADJ@linebreak.cjk@on}{%}
\@namedef{bbl@ADJ@linebreak.cjk@off}{%}
\@namedef{bbl@ADJ@justify.arabic@on}{%}
\@namedef{bbl@ADJ@justify.arabic@off}{%}
\@namedef{bbl@ADJ@layout.tabular@on}{%}
\@namedef{bbl@ADJ@layout.tabular@off}{%}
\@namedef{bbl@ADJ@layout#1@#2}{%
  \ifvmode
    \bbl@forkv{#1}{%
      %\bbl@ifunset{bbl@ADJ@##1@##2}%
      %{\bbl@cs{ADJ@##1}{##2}}%
      %{\bbl@cs{ADJ@##1@##2}}}}
  \fi%

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.

```latex
\DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
\DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
\DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
\DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
\DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
```

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.

```latex
\bbl@trace{Cross referencing macros}
\ifx\bbl@opt@safe\@empty\else % ie, if ‘ref’ and/or ‘bib’
  \def\newlabel#1#2#3{%
    \relax
    \if\@safe@activestrue
      \gdef\@multiplelabels{%
        \latex@warning@no@line{There were multiply-defined labels}}%
      \latex@warning@no@line{Label `#2' multiply defined}}%
    \global@namedef{#1@#2}{#3}}
\fi
```

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

```latex
\CheckCommand*\testdef[3] {%
  \if\reserved@a{#3}%
    \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
    \def\bbl@tempb{#3}%
  \else
    \@tempswatrue
  \fi
  \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
  \if\bbl@tempa\bbl@tempb
  \else
    \@tempswatrue
  \fi

\enddefault#1#2#3% TODO. With @samestring?
```

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.
\@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.

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

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

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

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

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

\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.

5.2 Marks

\markright This macro is used to pass the current language attribute to the headline routine. 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.

\bbl@trace{Marks}
\IfBabelLayout{sectioning}
{\ifx\bbl@opt@headfoot\@nnil
\g@addto@macro\@resetactivechars{\set@typeset@protect\select@language@x\expandafter{\bbl@main@language}}\let\protect\noexpand\edef\thepage{\noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}\fi}
{\ifbbl@single\bbl@ifunset{markright }\bbl@redefine\bbl@redefinerobust\markright#1{\bbl@ifblank{#1}{}\let\org@nocite\nocite\let\org@citex\@citex\let\org@bibcite\bibcite\let\org@bibitem\@bibitem\fi}

5.2 Marks

\markright This macro is used to pass the current language attribute to the headline routine. 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.

\bbl@trace{Marks}
\IfBabelLayout{sectioning}
{\ifx\bbl@opt@headfoot\@nnil
\g@addto@macro\@resetactivechars{\set@typeset@protect\select@language@x\expandafter{\bbl@main@language}}\let\protect\noexpand\edef\thepage{\noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}\fi}
{\ifbbl@single\bbl@ifunset{markright }\bbl@redefine\bbl@redefinerobust\markright#1{\bbl@ifblank{#1}{}\let\org@nocite\nocite\let\org@citex\@citex\let\org@bibcite\bibcite\let\org@bibitem\@bibitem\fi}
Thedefinitionof\markbothisequivalenttothatof\markright,exceptthatweneedtwo token registers. The document classes report and book define and set the headings for the page. While doing so they also store a copy of \markboth in \@mkboth. Therefore we need to check whether \@mkboth has already been set. If so we need to do that again with the new definition of \markboth. (As of Oct 2019, \LaTeX stores the definition in an intermediate macro, so it's not necessary anymore, but it's preserved for older versions.)

\ifx\@mkboth\markboth
\def\bbl@tempc{\let\@mkboth\markboth}
\else
\def\bbl@tempc{}\fi
\bbl@ifunset{markboth }\bbl@redefine\bbl@redefinerobust
\markboth#1#2{\protected@edef\bbl@tempb##1{\protect\foreignlanguage{\languagename}{\protect\bbl@restore@actives##1}}\bbl@ifblank{#1}{\toks@{}}{\toks\expandafter{\bbl@tempb{#1}}}\bbl@ifblank{#2}{\@temptokena{}}{\@temptokena\expandafter{\bbl@tempb{#2}}}\bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}}\bbl@tempc

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

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.

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.

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.

Deprecated. Use the tools provided by BgX. 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.
5.4 Encoding and fonts

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

The first thing we need to do is to determine, at `\begin{document}`, which Latin font encoding to use.

\texttt{\textbackslash 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.

\texttt{\textbackslash 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 `\if@packageloaded`) is disabled for this package. Now we have to revert to parsing the internal macro `\@filelist` which contains all the filenames loaded.

\texttt{\textbackslash AtBeginDocument{\ifx\UTFencname\@undefined}EU\ifcase\bbl@engine\or2\or1\fi} \texttt{\else} \texttt{\UTFencname} \texttt{\fi}\}

\texttt{\textbackslash latinencoding{\OT1}}\}
\texttt{\textbackslash cf@encoding\bbl@t@one} \texttt{\else} \texttt{\def\@elt#1{,#1,}} \texttt{\edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}} \texttt{\let\@elt\relax} \texttt{\bbl@xin{,T1,}\bbl@tempa} \texttt{\ifin\}} \texttt{\textbackslash latinencoding{\bbl@t@one}} \texttt{\fi}

\texttt{\fi}}\}

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

\texttt{\DeclareRobustCommand{\latintext}{\fontencoding{\latinencoding}\selectfont}} \texttt{\def\encodingdefault{\latinencoding}}

\texttt{\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.

\texttt{\AtBeginDocument{\ifx\@undefined\DeclareTextFontCommand}} \texttt{\DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}}} \texttt{\else} \texttt{\DeclareTextFontCommand{\textlatin}{\latintext}} \texttt{\fi}

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

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

### 5.5 Basic bidi support

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

It is loosely based on `rbabel.def`, but most of it has been developed from scratch. This babel module (by Johannes Braams and Boris Lavva) has served the purpose of typesetting R documents for two decades, and despite its flaws I think it is still a good starting point (some parts have been copied here almost verbatim), partly thanks to its simplicity. I’ve also looked at ARABI (by Youssef Jabri), which is compatible with babel.
There are two ways of modifying macros to make them “bidi”, namely, by patching the internal low-level macros (which is what I have done with lists, columns, counters, tocs, much like r\texttt{babel} 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 \texttt{\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\texttt{\TeX}-ja shows, vertical typesetting is possible, too.

\begin{verbatim}
4025 \bbl@trace{Loading basic (internal) bidi support}
4026 \ifodd\bbl@engine
4027 \else % TODO. Move to txt babel
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
4039 \def\bbl@loadxebidi#1{%
4040 \ifx\RTLfootnotetext\@undefined
4041 \AtEndOfPackage{%
4042 \EnableBabelHook{babel-bidi}%
4043 \bbl@loadfontspec % bidi needs fontspec
4044 \usepackage{bidi}%
4045 \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
4046 \def\DigitsDotDashInterCharToks{% See the 'bidi' package
4047 \ifnum\@nameuse{bbl@wdir@\languagename}=\tw@ % 'AL' bidi
4048 \bbl@digitsdotdash % So ignore in 'R' bidi
4049 \fi}%
4050 \fi}
4051 \ifnum\bbl@bidimode=200 % Any xe bidi=
4052 \ifcase\expandafter@\gobbletow\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=\one % 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
\end{verbatim}
Now the engine-dependent macros. TODO. Must be moved to the engine files.
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).

A tool for weak \textsc{I} (mainly digits). We also disable warnings with hyperref.

```latex
\DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
```
5.6 Local Language Configuration

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.

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).

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.
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.

Recognizing global options in packages not having a closed set of them is not trivial, as for them to be processed they must be defined explicitly. So, package options not yet taken into account and stored in bbl@language@opts are assumed to be languages. If not declared above, then 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.

```
\ifx\bbl@opt@main\@nnil
  \ifnum\bbl@iniflag>\z@ % if all ldf’s: set implicitly, no main pass
    \let\bbl@tempb\@empty
    \edef\bbl@tempsp{\classoptionslist,\bbl@language@opts}%
    \bbl@foreach\bbl@tempsp{% \bbl@tempsp is a reversed list
      \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
        \ifodd\bbl@iniflag % *=
          \IfFileExists{\bbl@opt@main@@ldf}{\def\bbl@opt@main{\bbl@opt@main}}{}
        \else % n +=
          \IfFileExists{\bbl@opt@main@@ldf}{\def\bbl@opt@main{\bbl@opt@main}}{}
        \fi
      \fi
    }%}
  \else
    \bbl@info{Main language set with 'main='. Except if you have\%
problems, prefer the default mechanism for setting\%
```

A few languages are still defined explicitly. They are stored in case they are needed in the 'main' pass (the value can be \relax).

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.

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

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

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 so afterBabelLanguage.
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.

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

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

6 The kernel of Babel (`babel.def`, common)
in a number of places. Some of the code below is common to plain \TeX and \TeX\, some of it is for the \TeX\ case only.
Plain formats based on etex (etex, extex, luaex) 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

\begin{verbatim}
\let\bbl@onlyswitch\@empty
\input babel.def
\let\bbl@onlyswitch\@undefined
\end{verbatim}

\section{Loading hyphenation patterns}

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

\begin{verbatim}
\@empty\toks@{ }
\def\bbl@languages{}
\end{verbatim}

\subsection{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.

\begin{verbatim}
\def\process@line#1#2 #3 #4 {%
  \ifx=#1%
    \process@synonym{#2}%
  \else
    \process@language{#1#2}{#3}{#4}%
  \fi
  \ignorespaces}
\end{verbatim}

\subsection{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.

\begin{verbatim}
\toks@{}
\def\bbl@languages{}
\end{verbatim}

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.

\begin{verbatim}
\def\process@synonym#1{%
  \ifnum\last@language=\m@ne
    \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
  \else
    \expandafter\chardef\csname l@#1\endcsname\last@language
    \wlog{\string\l@#1=\string\language\the\last@language}%
    \expandafter\let\csname #1hyphenmins\expandafter\endcsname
      \csname\languagename hyphenmins\endcsname
    \let\bbl@elt\relax
    \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}{}}%
  \fi
}\end{verbatim}

94
The macro \process@language is used to process a non-empty line from the ‘configuration file’. It has three arguments, each delimited by white space. The first argument is the ‘name’ of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions.

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

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

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

\begin{verbatim}
\def\process@language#1#2#3{%  \
\expandafter\addlanguage\csname l@#1\endcsname \expandafter\language\csname l@#1\endcsname \edef\languagename{#1}% \bbl@hook@everylanguage{#1}% % > luatex \bbl@get@enc#1::\@@@
\begingroup \lefthyphenmin\m@ne \bbl@hook@loadpatterns{#2}% % > luatex \ifnum\lefthyphenmin\m@ne \else \expandafter\xdef\csname #1hyphenmins\endcsname{\the\lefthyphenmin\the\righthyphenmin}\fi \endgroup \def\bbl@tempa{#3}% \ifx\bbl@tempa\@empty\else \bbl@hook@loadexceptions{#3}\fi % > luatex \let\bbl@elt\relax \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}\ifnum\the\language=\z@
\expandafter\ifx\csname #1hyphenmins\endcsname\relax \set@hyphenmins\tw@\thr@@\relax \else \expandafter\expandafter\expandafter\set@hyphenmins\csname #1hyphenmins\endcsname\fi \the\toks@\toks@{}\fi % > luatex \let\bbl@elt\relax \edef\bbl@languages{% \bbl@languages\bbl@elt{\the\language}{\#2}{\bbl@tempa}}\ifnum\the\language=\z@
\expandafter\expandafter\expandafter\set@hyphenmins\csname #1hyphenmins\endcsname\fi \the\toks@\toks@{}}% \fi
\end{verbatim}
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.

\begin{verbatim}
\def\bbl@get@enc#1:#2:#3@@@{
\def\bbl@hyph@enc{#2}}
\end{verbatim}

Now, hooks are defined. For efficiency reasons, they are dealt here in a special way. Besides \texttt{luatex}, format-specific configuration files are taken into account. \texttt{loadkernel} currently loads nothing, but define some basic macros instead.

\begin{verbatim}
\def\bbl@hook@everylanguage#1{}
\def\bbl@hook@loadpatterns#1{
\input #1 \relax}
\let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
\def\addlanguage{\csname newlanguage\endcsname}
\def\adddialect##1##2{
\global\chardef##1##2 \relax
\wlog{\string##1 = a dialect from \string\language##2}}
\def\iflanguage##1{
\expandafter\ifx\csname l@##1\endcsname\relax
\@nolanerr{##1}\else
\expandafter\expandafter\expandafter\@firstoftwo
\else
\expandafter\expandafter\expandafter\@secondoftwo
\fi}
\def\providehyphenmins##1##2{
\expandafter\ifx\csname ##1hyphenmins\endcsname\relax
\@namedef{##1hyphenmins}{##2}\fi}
\def\set@hyphenmins##1##2{
\lefthyphenmin##1 \relax
\righthyphenmin##2 \relax}
\def\selectlanguage{\errhelp{Selecting a language requires a package supporting it}\errmessage{Not loaded}}
\let\foreignlanguage\selectlanguage
\let\otherlanguage\selectlanguage
\let\otherlanguage*\selectlanguage
\def\bbl@usehooks##1##2{}% TODO. Temporary!!
\def\setlocale{\errhelp{Find an armchair, sit down and wait}\errmessage{Not yet available}}
\let\uselocale\setlocale
\let\locale\setlocale
\let\selectlocale\setlocale
\let\localename\setlocale
\let\textlocale\setlocale
\let\textlanguage\setlocale
\let\languagetext\setlocale
\begingroup
\def\AddBabelHook#1#2{
\expandafter\ifx\csname bbl@hook@#2\endcsname\relax
\def\next{\toks1}\else
\def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}\fi
\next}
\ifx\directlua\@undefined
\ifx\XeTeXinputencoding\@undefined\else
\input xebabel.def\fi
\else
\input luababel.def\fi
\endgroup
\end{verbatim}

96
The configuration file can now be opened for reading.

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

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 $-1$.

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.

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.

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.

We add a message about the fact that babel is loaded in the format and with which language patterns to the `\everyjob` register.
Also remove some macros from memory and raise an error if \toks0 is not empty. Finally load switch.def, but the latter is not required and the line inputting it may be commented out.

```
\let\bbl@line\@undefined
\let\process@line\@undefined
\let\process@synonym\@undefined
\let\process@language\@undefined
\let\bbl@get@enc\@undefined
\let\bbl@hyph@enc\@undefined
\let\bbl@tempa\@undefined
\let\bbl@hook@loadkernel\@undefined
\let\bbl@hook@everylanguage\@undefined
\let\bbl@hook@loadpatterns\@undefined
\let\bbl@hook@loadexceptions\@undefined
\langle/patterns\rangle
```

Here the code for ini\TeX ends.

## 8 Font handling with fontspec

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

```
\langle More package options\rangle ≡
\chardef\bbl@bidimode\z@% 4575
\DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
\DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }% 4576
\DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }%
\DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
\DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
\DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
\langle More package options\rangle
```

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 useless) message.

```
\langle Font selection\rangle ≡
\bbl@trace{Font handling with fontspec}
\ifx\EXPSyntaxOn\@undefined\else
  \def\bbl@fs@warn@nx#1#2{% \bbl@tempfs is the original macro
    \in@{{#1,},{no-script,language-not-exist,}\
    \ifin@\else\bbl@tempfs@nx{#1}{#2}\fi}
  }
  \def\bbl@fs@warn@nxx#1#2#3{% 
    \in@{{#1,},{no-script,language-not-exist,}\
    \ifin@\else\bbl@tempfs@nxx{#1}{#2}{#3}\fi}
  }
  \def\bbl@loadfontspec{\relax
    \if\fontspec\@undefined\else
      \usepackage{fontspec}\relax
    \fi}
\fi
```
\bbl@loadfontspec
\EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
\bbl@font
\newcommand\bbl@font[2][]{% 1=features 2=fontname, @font=rm|sf|tt
\bbl@ifunset{\bbl@tempb family}%
\bbl@providefam{\bbl@tempb}
}

% For the default font, just in case:
\bbl@ifunset{\bbl@lsys\languagename}{\bbl@provide\lsys\languagename}
\expandafter\bbl@ifblank\expandafter{\bbl@tempa}
\bbl@csarg\edef{\bbl@tempb dflt@}{\bbl@rmdflt@}
\bbl@foreach\bbl@font@fams{\bbl@tempa}{% ie bbl@rmdflt@lang / *scrt
\bbl@csarg\def{\bbl@tempb dflt@}{\bbl@rmdflt@}{\bbl@tempa})}

If the family in the previous command does not exist, it must be defined. Here is how:
\def\bbl@providefam#1{%
\bbl@exp{%\newcommand{\#1default}{}% Just define it
\bbl@add@list\bbl@font@fams{\#1}
\DeclareRobustCommand{\#1family}{%\not@math@alphabet\#1family}\relax
% \prepare@family@series@update{\#1}\#1default% TODO. Fails
\fontfamily{\#1default}\ifx>\UseHooks\@undefined\else\UseHook{\#1family}\fi%
\selectfont}
\DeclareTextFontCommand{\text\#1}{\#1family}}
The following macro is activated when the hook babel-fontspec is enabled. But before, we define a macro for a warning, which sets a flag to avoid duplicate them.
\def\bbl@nostdfont#1{%
\bbl@ifnull{\gdef{WFF@f@family}{}% Flag, to avoid dupl warns
\bbl@infowarn{The current font is not a babel standard family:\%
\fontname\font\%
There is nothing intrinsically wrong with this warning, and\%
you can ignore it altogether if you do not need these\%
families. But if they are used in the document, you should be\%
aware 'babel' will not set Script and Language for them, so\%
you may consider defining a new family with \string\babelfont.\%
See the manual for further details about \string\babelfont.\%
Reported}}
}
\gdef\bbl@switchfont{%
\bbl@ifnull{\bbl@lsys\languagename}{\bbl@provide\lsys\languagename}
\bbl@exp{\lowercase{\edef{\bbl@tempa}{\bbl@cl{sname}}}}
\bbl@foreach\bbl@font@fams{\bbl@tempa}{% (1) language?
\bbl@csarg\edef{\bbl@rmdflt@}{\bbl@tempa}{\bbl@tempname}{\bbl@rmdflt@}{\bbl@languagename}{\bbl@rmdflt@}{\bbl@tempa})}{% (2) from script?
\bbl@csarg\edef{\bbl@rmdflt@}{\bbl@tempa}{\bbl@tempname}{\bbl@rmdflt@}{\bbl@languagename}{\bbl@rmdflt@}{\bbl@tempa})}{% (3) from generic?
\bbl@exp{\global\let{\bbl@rmdflt@}{\bbl@languagename}{\bbl@rmdflt@}{\bbl@tempa})}{% 3=T - from generic
\bbl@exp{\global\let{\bbl@rmdflt@}{\bbl@languagename}{\bbl@rmdflt@}{\bbl@tempa})}{% 2=T - from script
\bbl@exp{\global\let{\bbl@rmdflt@}{\bbl@languagename}{\bbl@rmdflt@}{\bbl@tempa})}{% 1=T - language, already defined
\def\bbl@tempa{\bbl@nostdfont}{% TODO. Don’t use \bbl@tempa
\foreach\fontof\fams{% don't gather with prev for
  \ifunset{\famrst\#1}{%
    \exp{% order is relevant. TODO: but sometimes wrong!
      \add\originalTeX{%
        \fontrst{\cl\#1default}{\languagename}{\family}{\default}{\family}{\fontname}{font}}% 
      \fontset{\languagename}{\default}{\family}{\fontname} % the main part!
    }%
  }%
  }%
\ifrestoring{}{\temp}%

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

\ifx\f@family\@undefined\else % if latex
  \ifcase\bbl@engine % if pdftex
    \let\bbl@ckeckstdfonts\relax
  \else
    \def\bbl@ckeckstdfonts{%
      \begingroup
        \global\let\bbl@ckeckstdfonts\relax
        \let\temp\@empty
        \foreach\fontof\fams{%
          \ifunset{\#1default}{%
            \nameuse{\family} % Flag
            \exp{%
              \add\temp{%
                \family\=\f@family\\\%
                \space\fontname\font\\}
            }%
            \xdef{\#1default}{\f@family}
          }%
        }%
      \fi
      \endgroup
    }
  \fi
\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, \LaTeX\ 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*).

\def\bbl@fontset#1#2#3{% eg \bbl@rmdefault \rmfamily
  % bg
  \bbl@xin{<>}{#1}%
  \ifin%
  \exp{\fontset\#1{\#2}{\#3}}%
  \fi
  \exp{% 'Unprotected' macros return prev values
    \def\#2{\#1} % eg, \rmdefault{\bbl@rmdefault{\lang}}
  }%
  \ifsamestring{\#2}{\f@family}{
    \fontset{\#2}{\family}{\lang}
  }%
}
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.
Now, the code.

```latex
\def\BabelFootnote#1#2#3#4{\ifx\bbl@fn@footnote\@undefined\let\bbl@fn@footnote\footnote\fi\ifx\bbl@fn@footnotetext\@undefined\let\bbl@fn@footnotetext\footnotetext\fi\bbl@ifblank{#2}{\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}\@namedef{\bbl@stripslash#1text}{\bbl@footnotetext{\@firstofone}{#3}{#4}}}{{#1}{#2}{#3}{#4}}}}
```

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

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

\newcommand\babelinterchar[5][]{%
\let\bbl@kv@label\@empty
\bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}%
\@namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2@\empty}{%\ifnum\language=\l@nohyphenation\expandafter\@gobble\else\expandafter\@firstofone\fi{#5}}%
\bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone
{\bbl@exp{\bl@for\bbl@tempa{\zap@space#3@\empty}}{\bbl@exp{\bl@for\bbl@tempb{\zap@space#4@\empty}}{\XeTeXinterchartoks\@nameuse{bbl@xeclass@\bbl@tempa @\bbl@ifunset{bbl@xeclass@\bbl@tempa @#2}{}{#2}}%\@nameuse{bbl@xeclass@\bbl@tempb @\bbl@ifunset{bbl@xeclass@\bbl@tempb @#2}{}{#2}}%=\expandafter{\csname bbl@ic@\bbl@kv@label @#2\expandafter\endcsname\csname\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2@\empty@endcsname}}}}%
\DeclareRobustCommand\enablelocaleinterchar[1][]{%
\bbl@ifunset{bbl@ic@#1@\languagename}{\bbl@error{‘#1’ for ‘\languagename’ cannot be enabled.\%}{See the manual for further details.}}%
{\bbl@csarg\let{ic@#1@\languagename}\@gobble}}%
\DeclareRobustCommand\disablelocaleinterchar[1][]{%
\bbl@ifunset{bbl@ic@#1@\languagename}{\bbl@error{‘#1’ for ‘\languagename’ cannot be disabled.\%}{See the manual for further details.}}%
{\bbl@csarg\let{ic@#1@\languagename}@\gobble}}
⟨/xetex⟩

\section{Layout}

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titleps, and geometry. \bbl@startskip and \bbl@endskip are available to package authors. Thanks to the \TeX\ expansion
Consider `txt2latex` as a shorthand for `tex–xet latex`, which is the bidi model in both pdftex and xetex.
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.

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).

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

The names \texttt{\@\langle language\rangle} are defined and take some value from the beginning because all \texttt{.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 \texttt{ldf} finishes). If a language has been loaded, \texttt{\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 \texttt{language.dat} have the same name then just ignore the latter. If there are new synonymous, the 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 \texttt{luatex} patterns are best loaded when the document is typeset, and the "0th" language is preloaded just for backwards compatibility.

As of 1.1b, \texttt{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 \texttt{language.dat} is used (under the principle of a single source), instead of \texttt{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 cat code tables, but no format (targeted by babel) provide a command to allocate them (although there are packages like tablestack). 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 file is read at three places: (1) when \texttt{plain.def}, \texttt{babel.sty} starts, to read the list of available languages from \texttt{language.dat} (for the base option); (2) at \texttt{hyphen.cfg}, to modify some macros; (3) in the middle of \texttt{plain.def} and \texttt{babel.sty}, by \texttt{babel.def}, with the commands and other definitions for \texttt{luatex} (eg, \texttt{\bbl@patterns}).
\bbl@info{Non-standard hyphenation setup}\fi\let\bbl@manylang\relax\def\bbl@process@language#1#2#3{\ifcase\count@\@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}\or\count@\tw@\fi\ifnum\count@=\tw@\expandafter\addlanguage\csname l@#1\endcsname\language\allocationnumber\bbl@last\allocationnumber\bbl@manylang\let\bbl@elt\relax\xdef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}\the\toks@\toks@{}\fi\def\bbl@process@synonym@aux#1#2{\global\expandafter\chardef\csname l@#1\endcsname#2\relax\let\bbl@elt\relax\xdef\bbl@languages{\bbl@languages\bbl@elt{#1}{#2}{}{}}}\def\bbl@process@synonym#1{\ifcase\count@\toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}\or\@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}\else\bbl@process@synonym@aux{#1}{\the\bbl@last}\fi}\ifx\bbl@languages\@undefined\chardef\l@english\z@\chardef\l@USenglish\z@\chardef\bbl@last\z@\global\@namedef{bbl@hyphendata@0}{{hyphen.tex}{}}\gdef\bbl@languages{\bbl@elt{english}{0}{hyphen.tex}{}\bbl@elt{USenglish}{0}{}{}}\else\global\let\bbl@languages@format\bbl@languages\def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}\bbl@languages\openin\bbl@readstream=language.dat\ifeof\bbl@readstream\bbl@warning{I couldn't find language.dat. No additional\% patterns loaded. Reported}\else\loop\endlinechar\m@ne\read\bbl@readstream to \bbl@line\endlinechar\^^M\if T\ifeof\bbl@readstream\F\fi \relax\ifx\bbl@line\@empty\else\edef\bbl@line{\bbl@line\space\space\space}\fi\fi\xdef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}\fi\def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}\def\bbl@languages{\bbl@languages\bbl@elt{#1}{#2}{#3}{#4}}\def\bbl@elt{#1}{#2}{#3}{#4}}\def\bbl@process@synonym#1{\bbl@process@synonym#1\loop\endlinechar\m@ne\read\bbl@readstream to \bbl@line\endlinechar\^^M\if T\ifeof\bbl@readstream\F\fi \relax\ifx\bbl@line\@empty\else\edef\bbl@line{\bbl@line\space\space\space}\fi\fi}
function Babel.bytes(line)
  return line:gsub("(.)",
    function (chr) return unicode.utf8.char(string.byte(chr)) end)
end

function Babel.begin_process_input()
  if luatexbase and luatexbase.add_to_callback then
    luatexbase.add_to_callback('process_input_buffer',
      Babel.bytes,'Babel.bytes')
  else
    Babel.callback = callback.find('process_input_buffer')
    callback.register('process_input_buffer',Babel.callback)
  end
end

function Babel.end_process_input ()
  if luatexbase and luatexbase.remove_from_callback then
    luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
  else
    callback.register('process_input_buffer',Babel.callback)
  end
end

function Babel.addpatterns(pp, lg)
  local lg = lang.new(lg)
  local pats = lang.patterns(lg) or ''
  lang.clear_patterns(lg)
  for p in pp:gmatch('%[^%s]+') do
    ss = ''
    for i in string.utfcharacters(p:gsub('%d', '')) do
      ss = ss .. '%d?' .. i
    end
    ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
    ss = ss:gsub('%.%%d%?$', '%%."
    pats, n = pats:gsub('%%s .. ss .. '%s', ' ' .. p .. ' ') if n == 0 then
      tex.sprint( [[[string\csname bbl@info\endcsname{New pattern: }]]}

111
.. p .. [[]]])
pats = pats .. ' ' .. p
else
tex.sprint(
[\string\csname\space bbl@info\endcsname{Renew pattern: ]]
.. p .. [[]]])
end

... p ...

Babel.characters = Babel.characters or {}
Babel.ranges = Babel.ranges or {}
function Babel.hlist_has_bidi(head)
local has_bidi = false
local ranges = Babel.ranges
for item in node.traverse(head) do
  if item.id == node.id'glyph' then
    local itemchar = item.char
    local chardata = Babel.characters[itemchar]
    local dir = chardata and chardata.d or nil
    if not dir then
      for nn, et in ipairs(ranges) do
        if itemchar < et[1] then
          break
        elseif itemchar <= et[2] then
          dir = et[3]
          break
        end
      end
      if dir and (dir == 'al' or dir == 'r') then
        has_bidi = true
      end
    end
  end
  if dir and (dir == 'al' or dir == 'r') then
    has_bidi = true
  end
return has_bidi
end

function Babel.set_chranges_b (script, chrng)
  if chrng == '' then return end
texio.write('Replacing ' .. script .. ' script ranges')
Babel.script_blocks[script] = {}
for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
  table.insert(Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
end
end

function Babel.discard_sublr(str)
  if str:find( \\string\indexentry ) and
    str:find( \\string\babelsublr ) then
    str = str:gsub( \\string\babelsublr%s*(%b{})
      function(m) return m:sub(2,-2) end )
  end
return str
end


\iffalse newattribute\@undefined\else % Test for plain
\newattribute{bbl@attr@locale}
\directlua{ Babel.attr.locale = luatexbase.registernumber'bbl@attr@locale'}
\AddBabelHook{luatex}{beforeextras}{% \setattribute{bbl@attr@locale\localeid}
\fi
\def\BabelStringsDefault{unicode}
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.

\bbl@patterns
\AtEndOfPackage{\newcommand{\bbl@patterns}[2][\@empty]{\ifx\bbl@patterns@\relax\let\bbl@patterns@\@empty\fi\ifx\bbl@pttnlist\@empty\else\bbl@warning{You must not intermingle \string\selectlanguage\space and \string\bbl@patterns\space or some patterns will not\space be taken into account. Reported}\fi}}

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

10.4 Southeast Asian scripts
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.

\catcode`\%=14
\gdef\bbl@cjkintraspaces{}
\let\bbl@cjkintraspaces\relax
\directlua{
Babel = Babel or {}
Babel.cjk_enabled = true
function Babel.cjk_linebreak(head)
    local n = node.new(12, 13)  -- (glue, spaceskip)
    node.setglue(n, intraspace.b * quad,
                 intraspace.p * quad,
                 intraspace.m * quad)
    node.insert_before(head, item, n)
    node.remove(head, item)
end
end
end
end
^^
\bbl@luahyphenate}
local GLYPH = node.id 'glyph'
local last_char = nil
local quad = 655360 % 10 pt = 655360 = 10 * 65536
local last_class = nil
local last_lang = nil

for item in node.traverse(head) do
  if item.id == GLYPH then
    local lang = item.lang
    local LOCALE = node.get_attribute(item, 
      Babel.attr_locale)
    local props = Babel.locale_props[LOCALE]
    local class = Babel.cjk_class[item.char].c
    if props.cjk_quotes and props.cjk_quotes[item.char] then
      class = props.cjk_quotes[item.char]
    end
    if class == 'cp' then class = 'cl' end % )
    if class == 'id' then class = 'I' end
    local br = 0
    if class and last_class and Babel.cjk_breaks[last_class][class] then
      br = Babel.cjk_breaks[last_class][class]
    end
    if br == 1 and props.linebreak == 'c' and
      lang ~= \the\l@nohyphenation\space and
      last_lang ~= \the\l@nohyphenation then
      local n = node.new(14, 0) % penalty
      n.penalty = intrapenalty
      node.insert_before(head, item, n)
    end
    local intraspace = props.intraspace
    local n = node.new(12, 13) % (glue, spaceskip)
    node.setglue(n, intraspace.b * quad, 
      intraspace.p * quad, 
      intraspace.m * quad)
    node.insert_before(head, item, n)
  end
  if font.getfont(item.font) then
    quad = font.getfont(item.font).size
  end
  last_class = class
  last_lang = lang
  if penalty, glue or anything else
    last_class = nil
  end
end

lang.hyphenate(head)
end

\bbl@luahyphenate}
gdef\bbl@luahyphenate{%\let\bbl@luahyphenate\relax
\directlua{
luatexbase.add_to_callback('hyphenate',

116
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-
\begin{verbatim}
0634, 0635, 0636, 0637, 0639, 063A, 063B, 063C, 063D, 063E, 063F, %
0640, 0641, 0642, 0643, 0644, 0645, 0646, 0647, 0649}
\def\bblar@elongated{%
0626, 0628, 062A, 062B, 0633, 0634, 0635, 0636, 063B, %
063C, 063D, 063E, 063F, 0641, 0642, 0643, 0644, 0646, %
0649, 064A}
\begingroup
\catcode`_=11 \catcode`:=11
\gdef\bblar@nofswarn{\gdef\msg_warning:nnx##1##2##3{}}
\endgroup
\gdef\bbl@arabicjust{% TODO. Allow for several locales.
\let\bbl@arabicjust\relax
\newattribute\bblar@kashida
\directlua{ Babel.attr_kashida = luatexbase.registernumber\'bblar@kashida\' }
\bblar@kashida=\z@
\bbl@patchfont{{\bbl@parsejalt}}%
\directlua{
Babel.arabic.elong_map = Babel.arabic.elong_map or {}
Babel.arabic.elong_map\[\the\localeid\] = {}
luatexbase.add_to_callback\('post_linebreak_filter',
Babel.arabic.justify, 'Babel.arabic.justify')
luatexbase.add_to_callback\('hpack_filter',
Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
}}%
\gdef\bblar@fetchjalt#1#2#3#4{%
\bbl@foreach{#1}{%\bbl@ifunset{bblar@JE@##1}%
{\setbox\z@\hbox{\textdir TRT ^^^^200d\char"##1#2}}%
{\setbox\z@\hbox{\textdir TRT ^^^^200d\char"\@nameuse{bblar@JE@##1}#2}}%
\directlua{%
local last = nil
for item in node.traverse(tex.box[0].head) do
if item.id == node.id'glyph' and item.char > 0x600 and
not (item.char == 0x200D) then
last = item
end
end
Babel.arabic.#3\['##1#4'\] = last.char
}}%
\gdef\bblar@chars{^^^^064a}{from}{a}% Alef maksura
\gdef\bblar@chars{^^^^0649}{from}{y}% Yeh
\end{verbatim}
The actual justification (inspired by \textsc{chickenize}).

\begin{verbatim}
\directlua{
Babel.arabic = Babel.arabic or {}
Babel.arabic.from = {}
Babel.arabic.dest = {}
Babel.arabic.justify_factor = 0.95
Babel.arabic.justify_enabled = true
Babel.arabic.kashida_limit = -1

function Babel.arabic.justify(head)
  if not Babel.arabic.justify_enabled then return head end
  for line in node.traverse_id(node.id'\textit{hlist}', head) do
    Babel.arabic.justify_hlist(head, line)
  end
  return head
end

function Babel.arabic.justify_hbox(head, gc, size, pack)
  local has_inf = false
  if Babel.arabic.justify_enabled and pack == 'exactly' then
    for n in node.traverse_id(12, head) do
      if n.stretch_order > 0 then has_inf = true end
    end
    if not has_inf then
      Babel.arabic.justify_hlist(head, nil, gc, size, pack)
    end
  end
  return head
end

function Babel.arabic.justify_hlist(head, line, gc, size, pack)
  local d, new
  local k_list, k_item, pos_inline
  local width, width_new, full, k_curr, wt_pos, goal, shift
  local subst_done = false
  local elong_map = Babel.arabic.elong_map
  local cnt
  local last_line
  local GLYPH = node.id'\textit{glyph}'
  local KASHIDA = Babel.attr_kashida
  local LOCALE = Babel.attr_locale

  if line == nil then
    \endgroup}
\end{verbatim}
line = {}
line.glue_sign = 1
line.glue_order = 0
line.head = head
line.shift = 0
line.width = size
end

% Exclude last line. todo. But-- it discards one-word lines, too!
if (line.glue_sign == 1 and line.glue_order == 0) then
  elongss = {} % Stores elongated candidates of each line
  k_list = {} % And all letters with kashida
  pos_inline = 0 % Not yet used
for n in node.traverse_id(GLYPH, line.head) do
  pos_inline = pos_inline + 1 % To find where it is. Not used.
  if elong_map then
    local locale = node.get_attribute(n, LOCALE)
    if elong_map[locale] and elong_map[locale][n.font] and
      elong_map[locale][n.font][n.char] then
      table.insert(elongs, {node = n, locale = locale} )
      node.set_attribute(n.prev, KASHIDA, 0)
    end
  end
end % of node.traverse_id
if #elongs == 0 and #k_list == 0 then goto next_line end
full = line.width
shift = line.shift
goal = full * Babel.arabic.justify_factor % A bit crude
width = node.dimensions(line.head) % The 'natural' width

% == Elongated ==
% Original idea taken from 'chikenize'
while (#elongs > 0 and width < goal) do
  subst_done = true
  local x = #elongs
  local oldchar = curr.char
  curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
  width = node.dimensions(line.head) % Check if the line is too wide
  if width > goal then
    curr.char = oldchar
    break
  end
  table.remove(elongs, x)
end
% == Tatwil ==
if #k_list == 0 then goto next_line end

width = node.dimensions(line.head) % The 'natural' width
k_curr = #k_list % Traverse backwards, from the end
wt_pos = 1

while width < goal do
    subst_done = true
    k_item = k_list[k_curr].node
    if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
        d = node.copy(k_item)
        d.char = 0x0640
        d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
        d.xoffset = 0
        line.head, new = node.insert_after(line.head, k_item, d)
        width_new = node.dimensions(line.head)
        if width > goal or width == width_new then
            node.remove(line.head, new) % Better compute before
            break
        end
        Babel.fix_diacr then
            Babel.fix_diacr(k_item.next)
        end
    end
    if k_curr == 1 then
        k_curr = #k_list
        wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
    else
        k_curr = k_curr - 1
    end
end

% Limit the number of tatweel by removing them. Not very efficient,
% but it does the job in a quite predictable way.
if Babel.arabic.kashida_limit > -1 then
    cnt = 0
    for n in node.traverse_id(GLYPH, line.head) do
        if n.char == 0x0640 then
            cnt = cnt + 1
        if cnt > Babel.arabic.kashida_limit then
            node.remove(line.head, n)
        end
    end
else
    cnt = 0
end
end

::next_line::

% Must take into account marks and ins, see luatex manual.
% Have to be executed only if there are changes. Investigate
% what's going on exactly.
if subst_done and not gc then
    d = node.hpack(line.head, full, 'exactly')
    d.shift = shift
    node.insert_before(head, line, d)
    node.remove(head, line)
end
end % if process line
}}
10.7 Common stuff

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 function `Babel.locale_map`, which just traverses 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 each character to its 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 discretionary are handled in a special way.

% TODO - to a lua file
\directlua{
Babel.script_blocks = {
  ['dflt'] = {},
  ['Arab'] = {{0x0600, 0x06FF}, {0x0800, 0x08FF}, {0x0750, 0x077F},
               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
  ['Armn'] = {{0x0530, 0x058F}},
  ['Beng'] = {{0x0900, 0x09FF}},
  ['Cher'] = {{0x0C00, 0x0CFF}, {0x0B00, 0x0BFF}},
  ['Copt'] = {{0x0380, 0x03FF}, {0x0280, 0x02FF}, {0x1020, 0x10FF}},
  ['Cyril'] = {{0xD000, 0xD0FF}, {0x0500, 0x05FF}},
  ['Deva'] = {{0x0800, 0x08FF}},
  ['Ethi'] = {{0x0E00, 0x0EFF}, {0x0F00, 0x0FFF}},
  ['Geor'] = {{0x0A00, 0x0AFF}, {0x0D00, 0x0DFF}},
  ['Grek'] = {{0x0370, 0x03FF}, {0x0200, 0x02FF}, {0x1F00, 0x1FFF}},
  ['Hans'] = {{0x0E00, 0x0EFF}, {0x0F00, 0x0FFF}},
  ['Hebr'] = {{0x0D50, 0x0DFF}},
  ['Jpan'] = {{0x0100, 0x01FF}, {0x0080, 0x00FF}, {0x1E00, 0x1EFF},
               {0xD780, 0xD7FF}, {0xEF00, 0xEFFF}},
  ['Knda'] = {{0x0C80, 0x0CFF}},
  ['Kore'] = {{0x0E00, 0x0E9F}, {0x03F0, 0x03FF}, {0x1B00, 0x1BFF}},
  ['Khar'] = {{0x0C70, 0x0D7F}, {0x0C70, 0x0D7F}},
  ['Latn'] = {{0x0000, 0x00FF}, {0x0100, 0x01FF}, {0x0200, 0x02FF},
               {0x0A00, 0x0AFF}, {0x0B00, 0x0BFF}},
  ['Mlym'] = {{0x0D00, 0x0DFF}},
  ['Mahj'] = {{0x0110, 0x011F}},
  ['Mlym'] = {{0x0D00, 0x0DFF}},
  ['Myar'] = {{0x0D00, 0x0DFF}},
  ['Orya'] = {{0x0D00, 0x0DFF}},
  ['Sinh'] = {{0x0D00, 0x0DFF}},
  ['Syr'] = {{0x0D00, 0x0DFF}},
}
['Tam'] = {{0x0B80, 0x0BFF}},
['Tel'] = {{0x0C00, 0x0C7F}},
['Tfng'] = {{0x2D30, 0x2D7F}},
['Thai'] = {{0x0E00, 0x0E7F}},
['Tibt'] = {{0x0F00, 0x0FFF}},
['Vaii'] = {{0xA500, 0xA63F}},
['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
function Babel.locale_map(head)
if not Babel.locale_mapped then return head end
local LOCALE = Babel.attr_locale
local GLYPH = node.id('glyph')
local inmath = false
local toloc_save
for item in node.traverse(head) do
    local toloc
    if not inmath and item.id == GLYPH then
        % Optimization: build a table with the chars found
        if Babel.chr_to_loc[item.char] then
            toloc = Babel.chr_to_loc[item.char]
        else
            for lc, maps in pairs(Babel.loc_to_scr) do
                for _, rg in pairs(maps) do
                    if item.char >= rg[1] and item.char <= rg[2] then
                        Babel.chr_to_loc[item.char] = lc
                        toloc = lc
                        break
                    end
                end
            end
            if (item.char >= 0x0300 and item.char <= 0x036F) or
               (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
               (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
                Babel.chr_to_loc[item.char] = -2000
                toloc = -2000
            end
            if not toloc then
                Babel.chr_to_loc[item.char] = -1000
            end
        end
        if toloc == -2000 then
            toloc = toloc_save
        elseif toloc == -1000 then
            toloc = nil
        end
        if toloc and Babel.locale_props[toloc].letters and
           tex.getcatcode(item.char) \string= 11 then
            toloc = nil
        end
        if toloc and Babel.locale_props[toloc].script
           and Babel.locale_props[node.get_attribute(item, LOCALE)].script
           and Babel.locale_props[toloc].script ==
           Babel.locale_props[node.get_attribute(item, LOCALE)].script then
            toloc = nil
        end
    end
end
if toloc == -2000 then
tocol = toloc_save
elseif toloc == -1000 then
tocol = nil
end
if toloc and Babel.locale_props[toloc] and
   Babel.locale_props[toloc].letters and
   tex.getcatcode(item.char) \string= 11 then
    toloc = nil
end
if toloc and Babel.locale_props[toloc].script
   and Babel.locale_props[node.get_attribute(item, LOCALE)].script
   and Babel.locale_props[toloc].script ==
   Babel.locale_props[node.get_attribute(item, LOCALE)].script then
    toloc = nil
end
end
if toloc then
  if Babel.locale_props[toloc].lg then
    item.lang = Babel.locale_props[toloc].lg
    node.set_attribute(item, LOCALE, toloc)
  end
  if Babel.locale_props[toloc]['/'..item.font] then
    item.font = Babel.locale_props[toloc]['/'..item.font]
  end
  toloc_save = toloc
elseif not inmath and item.id == 7 then \% Apply recursively
  item.replace = item.replace and Babel.locale_map(item.replace)
  item.pre = item.pre and Babel.locale_map(item.pre)
  item.post = item.post and Babel.locale_map(item.post)
elseif item.id == node.id'math' then
  inmath = (item.subtype == 0)
end
return head
}
}

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

\newcommand\babelcharproperty[1]{\count@=#1\relax\ifvmode\expandafter\bbl@chprop\else\bbl@error{\string\babelcharproperty\space can be used only in vertical mode (preamble or between paragraphs)}{See the manual for further info}\fi}

\newcommand\bbl@chprop[3][\the\count@]{\@tempcnta=#1\relax\bbl@ifunset{bbl@chprop@#2}{\bbl@error{No property named '#2'. Allowed values are direction (bc), mirror (bmg), and linebreak (lb)}{See the manual for further info}}{}\loop\bbl@cs{chprop@#2}{#3}\ifnum\count@<\@tempcnta\advance\count@\@ne\repeat}

\def\bbl@chprop@direction#1{\directlua{Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}Babel.characters[\the\count@]['d'] = '\#1'}}

\let\bbl@chprop@bc=\bbl@chprop@direction
\def\bbl@chprop@mirror#1{\directlua{Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}Babel.characters[\the\count@]['m'] = '\number#1'}}

\let\bbl@chprop@bmg=\bbl@chprop@mirror
\def\bbl@chprop@linebreak#1{\directlua{Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}Babel.cjk_characters[\the\count@]['c'] = '\#1'}}
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.

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

\begin{verbatim}
\begingroup
\catcode`~=12
\catcode`%=12
\catcode`&=14
\catcode`|=12
\gdef\babelprehyphenation{&%
  \@ifnextchar[{{\bbl@settransform{0}}}{\bbl@settransform{0}[]}}
\gdef\babelposthyphenation{&%
  \@ifnextchar[{{\bbl@settransform{1}}}{\bbl@settransform{1}[]}}
\gdef\bbl@settransform#1[#2]#3#4#5{&%
  \ifcase#1
    \bbl@activateprehyphen
  or
  \bbl@activateposthyphen
  \fi
  \begingroup
  \def\babeltempa{\bbl@add@list\babeltempb}&%
  \let\babeltempb\@empty
  \def\bbl@tempa{#5}&%
  \bbl@replace\bbl@tempa{,}{ ,}&% TODO. Ugly trick to preserve {}
  \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
    \bbl@ifsamestring{##1}{remove}&%
    {\bbl@add@list\babeltempb{nil}}&%
    {\directlua{
      local rep = [=[##1]=]
      rep = rep:gsub('^%s*(remove)%s*[^%s,]*', 'remove = true')
      rep = rep:gsub('^%s*(insert)%s*[^%s,]*', 'insert = true')
      rep = rep:gsub('^%s*(string)%s*[^%s,]*', '[^%s,]*', Babel.capture_func)
      if #1 == 0 or #1 == 2 then
        rep = rep:gsub('^%s*(space)%s*[^%s,]*', '[^%s,]+%s+([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)%s+', 'space = { .. %2, %3, %4 .. }')
        rep = rep:gsub('^%s*(spacefactor)%s*[^%s,]*', '[^%s,]+%s+([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)%s+', 'spacefactor = { .. %2, %3, %4 .. }')
      end
      if #1 == 1 then
        rep = rep:gsub('^%s*(kashida)%s*[^%s,]*', '[^%s,]*', Babel.capture_kashida)
      else
        rep = rep:gsub('^%s*(no)%s*[^%s,]*', '[^%s,]*', Babel.capture_func)
        rep = rep:gsub('^%s*(pre)%s*[^%s,]*', '[^%s,]*', Babel.capture_func)
        rep = rep:gsub('^%s*(post)%s*[^%s,]*', '[^%s,]*', Babel.capture_func)
      end
    }}&%
  tex.print([[[\string\babeltemp{[]} .. rep .. []]])
  }}&%
\endgroup
\end{verbatim}
\foreach \bbl@tempb{&%
\bbl@forkv{{##1}}{&%
in@{,####1,}{,nil,step,data,remove,insert,string,no,pre,\%}
no,post,payention,kashida,space,spacefactor,\%}
\ifin@else
\bbl@error
{Bad option '####1' in a transform.\\%}
I’ll ignore it but expect more errors\\%
(See the manual for further info.)\\%
\fi}}&%
\let\bbl@kv@attribute\relax
\let\bbl@kv@label\relax
\let\bbl@kv@fonts\@empty
\bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}&%
\ifx\bbl@kv@fonts\@empty\else\bbl@settransfont\fi
\ifx\bbl@kv@fonts\@empty\else\bbl@settransfont\fi
\ifx\bbl@kv@attribute\relax
\ifx\bbl@kv@label\relax\else
\bbl@exp{\\bbl@trim@def\\bbl@kv@fonts{\bbl@kv@fonts}}&%
\bbl@replace\bbl@kv@fonts{ }{,}&%
\edef\bbl@kv@attribute{bbl@ATR@\bbl@kv@label @#3@\bbl@kv@fonts}&%
\count@\z@
\def\bbl@elt##1##2##3{&%
\bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
{\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
{\count@\@ne}&%
{\bbl@error
{Transforms cannot be re-assigned to different\%}
fonts. The conflict is in 'bl@kv@label'.\\%}
Apply the same fonts or use a different label\\%
(See the manual for further details.)}\\%}
{}}\\%}
\bbl@transfont@list
\ifnum\count@=\z@
\bbl@exp{\global\\bbl@add\bbl@transfont@list
{\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}\\%
\fi
\bbl@ifunset{\bbl@kv@attribute}&%
{\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
{}&%
\global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
\fi
\else
\edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}\\%
\fi
\directlua{local lbkr = Babel.linebreaking.replacements[#1]
local u = unicode.utf8
local id, attr, label
if #1 == 0 then
  id = \the\csname bbl@id@@#3\endcsname\space
else
  id = \the\csname l@#3\endcsname\space
end
\ifx\bbl@kv@attribute\relax
  attr = -1
\else
  attr = luatexbase.registernumber'\bbl@kv@attribute'
\fi
\ifx\bbl@kv@label\relax\else &% Same refs:
  label = [==[\bbl@kv@label]==]
\fi
&% Convert pattern:
local patt = string.gsub([==[#4]==], '%s', '')
if #1 == 0 then
    patt = string.gsub(patt, '|', ' ')
end
if not u.find(patt, '()', nil, true) then
    patt = '()' .. patt .. '()'
end
if #1 == 1 then
    patt = u.gsub(patt, '%%(%)%^', '^()')
    patt = string.gsub(patt, '%$%(%)', '()$')
end
patt = u.gsub(patt, '{(.)}', function (n)
    return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
end)
patt = u.gsub(patt, '{(%x%x%x%x+)}', function (n)
    return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
end)

lbkr[id] = lbkr[id] or {}
table.insert(lbkr[id], { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
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.

\newcommand{\localeprehyphenation}{[1]{\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.

\def{\bbl@activate@preotf}{\let{\bbl@activate@preotf}\relax % only once \directlua{ Babel = Babel or {} \function{Babel.pre_otfload_v}{head}{if Babel.numbers and Babel.digits_mapped then head = Babel.numbers(head) end if Babel.bidi enabled then head = Babel.bidi(head, false, dir) end return head} \function{Babel.pre_otfload_h}{head, gc, sz, pt, dir}{if Babel.numbers and Babel.digits_mapped then head = Babel.numbers(head) end if Babel.bidi enabled then head = Babel.bidi(head, false, dir) end return head} \luatexbase.add_to_callback('pre_linebreak_filter', \Babel.pre_otfload_v, \Babel.pre_otfload_v, \Babel.pre_otfload_v', \\Babel.pre_otfload_h, \\Babel.pre_otfload_h', \luatexbase.priority_in_callback('pre_linebreak_filter', \\luaotfload.node_processor') \nil)}

128
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=.

\def\bbl@insidemath{0}
\def\bbl@everymath{
  \def\bbl@insidemath{1}}
\def\bbl@everydisplay{
  \def\bbl@insidemath{2}}

\ifnum\bbl@bidimode>\z@ % Any bidi= except default=1
  \let\bbl@beforeforeign\leavevmode
  \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
  \RequirePackage{luatexbase}
  \bbl@activate@preotf
  \directlua{
    require('babel-data-bidi.lua')
    \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
      require('babel-bidi-basic.lua')\or
        require('babel-bidi-basic-r.lua')\fi}
  \newattribute\bbl@attr@dir
  \directlua{
    Babel.attr_dir = luatexbase.registernumber'\bbl@attr@dir' }
  \bbl@exp{\output{\bodydir\pagedir\the\output}}
  \fi
\fi
\chardef\bbl@thetextdir\z@
\chardef\bbl@thepardir\z@
\def\bbl@getluadir#1{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
  \ifcase\numexpr3-#1\relax
    \ifcase\bbl@getluadir{#1}\relax
      #2 TLT\relax
    \else
      #2 TRT\relax
    \fi
  \fi
  \def\bbl@thedir{0}
  \def\bbl@textdir#1{% Used twice
    \bbl@setluadir{text}\textdir{#1}%
    \chardef\bbl@thetextdir#1elax
    \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
    \setattribute{\bbl@attr@dir}{\numexpr\bbl@thepardir*4+#1}}
\def\bbl@pardir#1{% Used twice
  \bbl@setluadir{par}\pardir{#1}%
  \chardef\bbl@thepardir#1elax}
\def\bbl@bodydir{
  \bbl@setluadir{body}\bodydir%
  \chardef\bbl@bodydir\bodydir%

\ifnum\bbl@bidimode=20 % Any bidi= except default=1
  \def\bbl@insidemath{0}\
  \def\bbl@everymath{\def\bbl@insidemath{1}\
  \def\bbl@everydisplay{\def\bbl@insidemath{2}}

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.
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 \textit{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).

\texttt{\texttt{@hangfrom}} is useful in many contexts and it is redefined always with the \texttt{layout} option. There are, however, a number of issues when the text direction is not the same as the box direction (as set by \texttt{\bodydir}), and when \texttt{\parbox} and \texttt{\hangindent} are involved. Fortunately, latest releases of luatex simplify a lot the solution with \texttt{\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, \texttt{tabular} seems to work (at least in simple cases) with \texttt{array}, \texttt{tabularx}, \texttt{hhline}, \texttt{colortbl}, \texttt{longtable}, \texttt{booktabs}, etc. However, \texttt{dcolumn} still fails.

6493 \bbl@trace{Redefinitions for bidi \texttt{layout}}
6494 \%
6495 \{{"More package options}\} ⇒
6496 \chardef\bbl@eqnpos\z@
6497 \ DeclareOption{leqno}{\chardef\bbl@eqnpos@\@ne}
6498 \ DeclareOption{fleqn}{\chardef\bbl@eqnpos@\tw@}
6499 \{("More package options\}"
7400 \%
7401 \ifnum\bbl@bidimode=\z@ % Any bidi=
7402 \matheqdir\texttt{mode}@\@ne % A luatex primitive
7403 \let\bbl@eqnodir=\relax
7404 \def\bbl@eqdel{}
7405 \def\bbl@eqnum{%
7406 \normalfont\normalcolor
7407 \expandafter\@firstoftwo\bbl@eqdel
7408 \theequation
7409 \expandafter\@secondoftwo\bbl@eqdel}}
\def\bbl@provide@extra#1{% 
\fi
\fi
\fi
\fi
\fi
\def\bbl@provide@extra#1{% 
% == Counters: mapdigits ==
% Native digits
% If \bbl@KVP@mapdigits@nil
\def\bbl@ifunset{bbl@dgnat@languagename}{}%
% \RequirePackage{luatexbase}%
\bbl@activate@preotf
\directlua{

Babel = Babel or {} %%% \textaddress{presets in luabelabel}
Babel.digits_mapped = true
Babel.digits = Babel.digits or { }
Babel.digits[the\localeid] =
    \Babel\cl{dgnat}()
if not Babel.numbers then
    function Babel.numbers(head)
        local LOCALE = Babel.attr\locale
        local GLYPH = node\id{glyph}
        local inmath = false
        for item in node\traverse\(head\) do
            if not inmath and item\id == GLYPH then
                local temp = node\get\attribute\item,\locale
                if Babel.digits[temp] then
                    local chr = item\char
                    if chr > 47 and chr < 58 then
                        item\char = Babel.digits[temp][chr-47]
                    end
                end
            elseif item\id == node\id{math} then
                inmath = (item\subtype == 0)
            end
        return head
    end
end
\end
}%
\fi
% == transforms ==
% \if\bbl@KVP@transforms@nil
\def\bbl@elt##1##2##3{%
\in\$transforms.\$##1% 
\if\in%
\def\bbl@tempa{##1}%
\bbl@replace\bbl@tempa{transforms.}%
\bbl@carg\bbl@transforms{babel\bbl@tempa}{##2}{##3}%
\fi}%
\csname bbl@inidata@languagename\endcsname
\bbl@release@transforms\relax % \relax closes the last item.
\fi
% Start tabular here:
\def\localerestoredirs{% 
\ifcase\bbl@thetextdir
\ifnum\textdirection=\z@\else\textdir TLT\fi
\else
\ifnum\textdirection=\@ne\else\textdir TRT\fi
\fi
% \for\item\in\node\traverse\(head\) do
%     if not inmath and item\id == GLYPH then
%         local temp = node\get\attribute\item,\locale
%         if Babel.digits[temp] then
%             local chr = item\char
%             if chr > 47 and chr < 58 then
%                 item\char = Babel.digits[temp][chr-47]
%             end
%         end
%     elseif item\id == node\id{math} then
%         inmath = (item\subtype == 0)
%     end
% return head
% end
% end
% end
%\}%
\fi
% == transforms ==
% \if\bbl@KVP@transforms@nil
\def\bbl@elt##1##2##3{% 
\in\$transforms.\$##1% 
\if\in%
\def\bbl@tempa{##1}%
\bbl@replace\bbl@tempa{transforms.}%
\bbl@carg\bbl@transforms{babel\bbl@tempa}{##2}{##3}%
\fi}%
\csname bbl@inidata@languagename\endcsname
\bbl@release@transforms\relax % \relax closes the last item.
\fi
% Start tabular here:
\def\localerestoredirs{% 
\ifcase\bbl@thetextdir
\ifnum\textdirection=\z@\else\textdir TLT\fi
\else
\ifnum\textdirection=\@ne\else\textdir TRT\fi
\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.

\AtBeginDocument{%}
\ifpackageloaded{multicol}%
{\toks@\expandafter{\multi@column@out}%%
\edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
\fi
\AtBeginDocument{%}
\ifpackageloaded{paracol}%
{\edef\pcol@output{\bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
\fi

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. \bb@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.
\def\bbl@nextfake#1{% non-local changes, use always inside a group!
  \bbl@exp{% 
    \def\bbl@inside#1\bbl@nextfake\bbl@exp\bbl@inside{0}%% Once entered in math, set boxes to restore values
    \ifmmode
      \everyvbox{% 
        \the\everyvbox \bbl@insidemath{0} \mathdir \the\bodydir \everyhbox{\the\everyhbox}%
        \everyvbox{\the\everyvbox}}%
      \everyhbox{% 
        \the\everyhbox \bbl@insidemath{0} \mathdir \the\bodydir \everyhbox{\the\everyhbox}%
        \everyvbox{\the\everyvbox}}%
      \ifnum\bbl@getluadir{page}=-\bbl@getluadir{par}\else
        \shapemode@one
      \fi
    \fi
  }%}
\def\@hangfrom#1{% 
  \setbox\@tempboxa\hbox{{#1}}
  \hangindent\wd\@tempboxa
  \ifnum\bbl@getluadir{page}=-\bbl@getluadir{par}\else\shapemode@one\fi
  \noindent\box\@tempboxa}
\IfBabelLayout{tabular}{
  \let\bbl@OL@@tabular\@tabular
  \bbl@replace\@tabular{$}{\bbl@nextfake$}%
  \let\bbl@NL@@tabular\@tabular
  \AtBeginDocument{% 
    \ifx\bbl@NL@@tabular\@tabular
      \bbl@exp{\in@{\in\bbl@nextfake\@tabular}{\@tabular}}%
    \else
      \bbl@replace\@tabular{$}{\bbl@nextfake$}%
    \fi
    \let\bbl@NL@@tabular\@tabular
  }%}
\IfBabelLayout{lists}{
  \let\bbl@OL@list\list
  \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
  \let\bbl@NL@list\list
  \def\bbl@listparshape#1#2#3{\parshape #1 #2 #3 %
    \ifnum\bbl@getluadir{page}=-\bbl@getluadir{par}\else
      \ifnum\bbl@getluadir{page}=-\bbl@getluadir{par}\else
        \shapemode@two
      \fi
    \fi
  }%
}\IfBabelLayout{graphics}{
  \def\bbl@pictsetdir#1{\ifcase\bbl@thetextdir
    \let\bbl@pictresetdir\relax
  \else
    \ifcase#1\bodydir TLT % Remember this sets the inner boxes
      \or\textdir TLT
    \else\bodydir TLT \textdir TLT
  \fi
  \fi
  \bbl@pictresetdir\bodydir TLT \pardir TLT \textdir TLT \relax}\fi
}
\AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
\directlua{
Babel.get_picture_dir = true
Babel.picture_has_bidi = 0
%
function Babel.picture_dir (head)
  if not Babel.get_picture_dir then return head end
  if Babel.hlist_has_bidi(head) then
    Babel.picture_has_bidi = 1
  end
  return head
end
luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
  "Babel.picture_dir")%
\AtBeginDocument{%
  \def\LS@rot{%
    \setbox\@outputbox\vbox{%
      \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}}%
  \long\def\put(#1,#2)#3{%
    \@killglue%
    % Try:
    \ifx\bbl@pictresetdir\relax
      \def\bbl@tempc{0}%
    \else
      \directlua{
        Babel.get_picture_dir = true
        Babel.picture_has_bidi = 0
      }%
      \setbox\z@\hb@xt@\z@{%
        \@defaultunitsset\@tempdimc{#1}\unitlength
        \kern\@tempdimc
        \hss}% TODO: #3 executed twice (below). That's bad.
    \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
    \fi
  % Do:
    \@defaultunitsset\@tempdimc{#2}\unitlength
    \raise\@tempdimc\hb@xt@\z@{%
      \@defaultunitsset\@tempdimc{#1}\unitlength
      \kern\@tempdimc
      \{\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi\#3\}\hss}%
      \ignorespaces%
  \MakeRobust\put}%
  \AtBeginDocument{%
    \AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir\@gobble}%
    \ifx\pgfpicture@undefined\else % T000. Allow deactivate?
      \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\atne}%
      \bbl@add\pgfinterruptpicture{\bbl@pictsetdir}@%
    \fi
    \ifx\tikzpicture@undefined\else
      \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\tw@}%
      \bbl@add\tikz@atbegin@node{\bbl@pictsetdir}@%
      \bbl@sreplace\tikz{\begin{tikzpicture}\bbl@pictsetdir\tw@}%
    \fi
    \ifx\tcolorbox@undefined\else
      \def\tcb@drawing@env@begin{%
        \csname tcb@before@\tcb@split@state\endcsname
        \bbl@pictsetdir\tw@%
      \begin\{kvtcb@graphenv\}%
      \tcb@bbdraw%
      \tcb@apply@graph@patches%
    }%
Implicitly reverses sectioning labels in bidi=basic-r, because the full stop is not in contact with L numbers any more. I think there must be a better way. Assumes bidi=basic, but there are some additional readjustments for bidi=default.

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

\section{Lua: transforms}

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

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

> Babel.linebreaking.replacements = {}
> Babel.linebreaking.replacements[0] = {} -- pre
> Babel.linebreaking.replacements[1] = {} -- post

Discretionaries contain strings as nodes

function Babel.str_to_nodes(fn, matches, base)
    local n, head, last
    if fn == nil then return nil end
    for s in string.utfvalues(fn(matches)) do
        if base.id == 7 then
            base = base.replace
        end
        n = node.copy(base)
        n.char = s
        if not head then
            head = n
        else
            last.next = n
        end
        last = n
    end
    return head
end

Babel.fetch_subtext = {}

Babel.ignore_pre_char = function(node)
    return (node.lang == Babel.nohyphenation)
end

-- Merging both functions doesn't seem feasible, because there are too many differences.
Babel.fetch_subtext[0] = function(head)
    local word_string = ''
    local word_nodes = {}
    local lang
    local item = head
    local inmath = false
    while item do
        if item.id == 11 then
            inmath = (item.subtype == 0)
        end
        if inmath then
            -- pass
        elseif item.id == 29 then
            local locale = node.get_attribute(item, Babel.attr_locale)
            if lang == locale or lang == nil then
                lang = locale or locale
            end
        end
        item = item.next
    end
    return head
end
if Babel.ignore_pre_char(item) then
    word_string = word_string .. Babel.us_char
else
    word_string = word_string .. unicode.utf8.char(item.char)
end
word_nodes[#word_nodes+1] = item
else
    break
end
elseif item.id == 12 and item.subtype == 13 then
    word_string = word_string .. ''
    word_nodes[#word_nodes+1] = item
-- Ignore leading unrecognized nodes, too.
elseif word_string ~= '' then
    word_string = word_string .. Babel.us_char
    word_nodes[#word_nodes+1] = item -- Will be ignored
end
item = item.next
-- Here and above we remove some trailing chars but not the
-- corresponding nodes. But they aren't accessed.
if word_string:sub(-1) == ' ' then
    word_string = word_string:sub(1,-2)
end
word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
return word_string, word_nodes, item, lang
end
Babel.fetch_subtext[1] = function(head)
    local word_string = ''
    local word_nodes = {}
    local lang
    local item = head
    local inmath = false
    while item do
        if item.id == 11 then
            inmath = (item.subtype == 0)
        end
        if inmath then
            -- pass
        else
            if item.id == 7 and item.subtype == 2 then
                word_string = word_string .. '='
                word_nodes[#word_nodes+1] = item
            end
            else item.id == 29 then
                if item.lang == lang or lang == nil then
                    if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
                        lang = lang or item.lang
                        word_string = word_string .. unicode.utf8.char(item.char)
                        word_nodes[#word_nodes+1] = item
                    end
                else
                    break
                end
            else item.id == 7 and item.subtype == 2 then
                word_string = word_string .. '='
                word_nodes[#word_nodes+1] = item
            end
        end
    end
end
elseif item.id == 7 and item.subtype == 3 then
  word_string = word_string .. '|' 
  word_nodes[#word_nodes+1] = item

-- (1) Go to next word if nothing was found, and (2) implicitly 
-- remove leading USs.
elseif word_string == '' then
  -- pass
else
  word_string = word_string .. Babel.us_char 
  word_nodes[#word_nodes+1] = item -- Will be ignored
end
item = item.next

word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '*$', '')
return word_string, word_nodes, item, lang

function Babel.pre_hyphenate_replace(head)
  Babel.hyphenate_replace(head, 0)
end

function Babel.post_hyphenate_replace(head)
  Babel.hyphenate_replace(head, 1)
end

Babel.us_char = string.char(31)

function Babel.hyphenate_replace(head, mode)
  local u = unicode.utf8
  local lbkr = Babel.linebreaking.replacements[mode]
  local word_head = head

  while true do 
    -- for each subtext block
    local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
    if Babel.debug then
      print()
      print((mode == 0) and '@@@@<' or '@@@@>', w)
    end
    if nw == nil and w == '' then break end
    if not lang then goto next end
    if not lbkr[lang] then goto next end
    -- For each saved (pre|post)hyphenation. TODO. Reconsider how 
    -- loops are nested.
    for k=1, #lbkr[lang] do
      local p = lbkr[lang][k].pattern
      local r = lbkr[lang][k].replace
      local attr = lbkr[lang][k].attr or -1
      if Babel.debug then

print('*****', p, mode)

-- This variable is set in some cases below to the first *byte*
-- after the match, either as found by u.match (faster) or the
-- computed position based on sc if w has changed.
local last_match = 0
local step = 0

-- For every match.
while true do
  if Babel.debug then
    print('=====')
  end
  local new -- used when inserting and removing nodes

  local matches = { u.match(w, p, last_match) }
  if #matches < 2 then break end

  local first = table.remove(matches, 1)
  local last = table.remove(matches, #matches)
  if string.find(w:sub(first, last-1), Babel.us_char) then break end

  local save_last = last -- with A()BC()D, points to D

  -- Fix offsets, from bytes to unicode. Explained above.
  first = u.len(w:sub(1, first-1)) + 1
  last = u.len(w:sub(1, last-1)) -- now last points to C

  local sc = first-1 -- Used below, too

  local data_nodes = {}
  local enabled = true
  for q = 1, last-first+1 do
    data_nodes[q] = w_nodes[sc+q]
    if enabled
      and attr > -1
      and not node.has_attribute(data_nodes[q], attr)
    then
      enabled = false
    end
  end

  -- This loop traverses the matched substring and takes the
  -- corresponding action stored in the replacement list.
  -- sc = the position in substr nodes / string
  -- rc = the replacement table index
  local rc = 0
  while rc < last-first+1 do -- for each replacement
    if Babel.debug then
      print('......', rc + 1)
    end
  end
sc = sc + 1
rc = rc + 1

if Babel.debug then
Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
local ss = ''
for itt in node.traverse(head) do
  if itt.id == 29 then
    ss = ss .. unicode.utf8.char(itt.char)
  else
    ss = ss .. '(' .. itt.id .. ')
  end
end
print('***************', ss)
end

local crep = r[rc]
local item = w_nodes[sc]
local item_base = item
local placeholder = Babel.us_char
local d

if crep and crep.data then
  item_base = data_nodes[crep.data]
end

if crep then
  step = crep.step or 0
end

if (not enabled) or (crep and next(crep) == nil) then -- = {}
  last_match = save_last -- Optimization
  goto next
end

elseif crep == nil or crep.remove then
  node.remove(head, item)
  table.remove(w_nodes, sc)
  w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
  sc = sc - 1 -- Nothing has been inserted.
  last_match = utf8.offset(w, sc+1+step)
  goto next
end

elseif crep and crep.kashida then -- Experimental
  node.set_attribute(item,
    Babel.attr_kashida,
    crep.kashida)
  last_match = utf8.offset(w, sc+1+step)
  goto next
end

elseif crep and crep.string then
  local str = crep.string(matches)
  if str == '' then -- Gather with nil
    node.remove(head, item)
    table.remove(w_nodes, sc)
    w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
    sc = sc - 1 -- Nothing has been inserted.
  else
    local loop_first = true
    for s in string.utfvalues(str) do
      d = node.copy(item_base)
      d.char = s
      if loop_first then
        node.set_attribute(d, Babel.attr_kashida, crep.kashida)
        last_match = utf8.offset(w, sc+1+step)
        goto next
      end
      node.insert(head, d)
      table.insert(w_nodes, sc, d)
      w = w .. s
      sc = sc + 1
    end
    last_match = utf8.offset(w, sc+1+step)
    goto next
  end
end
loop_first = false

head, new = node.insert_before(head, item, d)

if sc == 1 then
    word_head = head
end

w_nodes[sc] = d

w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
else
    sc = sc + 1

head, new = node.insert_before(head, item, d)
table.insert(w_nodes, sc, new)

w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
end

if Babel.debug then
    print('.....', 'str')
    Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
end
end -- for

node.remove(head, item)
end -- if ''

goto next

elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
    d = node.new(7, 3) -- (disc, regular)
    d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
    d.post = Babel.str_to_nodes(crep.post, matches, item_base)
    d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
    d.attr = item_base.attr
    if crep.pre == nil then -- TeXbook p96
        d.penalty = crep.penalty or tex.hyphenpenalty
    else
        d.penalty = crep.penalty or tex.exhyphenpenalty
    end
    placeholder = '
'
    head, new = node.insert_before(head, item, d)
elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
    -- ERROR
elseif crep and crep.penalty then
    d = node.new(14, 0) -- (penalty, userpenalty)
    d.attr = item_base.attr
    d.penalty = crep.penalty
    head, new = node.insert_before(head, item, d)
elseif crep and crep.space then
    -- 655360 = 10 pt = 10 * 65536 sp
    d = node.new(12, 13) -- (glue, space)
    local quad = font.getfont(item_base.font).size or 655360
    node.setglue(d, crep.space[1] * quad,
                 crep.space[2] * quad,
                 crep.space[3] * quad)
    if mode == 0 then
        placeholder = ' '
    end
    head, new = node.insert_before(head, item, d)
elseif crep and crep.spacefactor then
    d = node.new(12, 13) -- (glue, space)
    local base_font = font.getfont(item_base.font)
    node.setglue(d,
                 crep.spacefactor[1] * base_font.parameters['space'],
                 ...)
crep.spacefactor[2] * base_font.parameters['space_stretch'],
crep.spacefactor[3] * base_font.parameters['space_shrink'])
if mode == 0 then
    placeholder = ''
end
head, new = node.insert_before(head, item, d)
elseif mode == 0 and crep and crep.space then
    -- ERROR
end -- ie replacement cases
-- Shared by disc, space and penalty.
if sc == 1 then
    word_head = head
end
if crep.insert then
    w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
    table.insert(w_nodes, sc, new)
    last = last + 1
else
    w_nodes[sc] = d
    node.remove(head, item)
    w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
end
last_match = utf8.offset(w, sc+1+step)
...next::
end -- for each replacement
if Babel.debug then
    print('.....', '/')
    Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
end
end -- for match
end -- for patterns
...next::
word_head = nw
end -- for substring
return head
-- This table stores capture maps, numbered consecutively
Babel.capture_maps = {}
-- The following functions belong to the next macro
function Babel.capture_func(key, cap)
    local ret = "[[".. cap.gsub('{{[0-9]}}', '', m[1]..[" '.. "]")
    local cnt
    local u = unicode.utf8
    ret, cnt = ret.gsub('{{[0-9]}}|{{\|}}|{{\)}}', Babel.capture_func_map)
    if cnt == 0 then
        ret = u.gsub(ret, '%x%x%x%x+}
        function (n)
            return u.char(tonumber(n, 16))
        end)
    end
    ret = ret.gsub('%[%]%%.%', '')
function Babel.capt_map(from, mapno)
  return Babel.capture_maps[mapno][from] or from
end

-- Handle the {n|abc|ABC} syntax in captures
function Babel.capture_func_map(capno, from, to)
  local u = unicode.utf8
  from = u.gsub(from, '{(\%x\%x\%x+)\}', function (n)
    return u.char(tonumber(n, 16))
  end)
  to = u.gsub(to, '{(\%x\%x\%x+)\}', function (n)
    return u.char(tonumber(n, 16))
  end)
  local froms = {}
  for s in string.utfcharacters(from) do
    table.insert(froms, s)
  end
  local cnt = 1
  table.insert(Babel.capture_maps, {})  
  local mlen = table.getn(Babel.capture_maps)
  for s in string.utfcharacters(to) do
    Babel.capture_maps[mlen][froms[cnt]] = s
    cnt = cnt + 1
  end
  return "]]..Babel.capt_map(m[" .. capno .. "]," ..
     (mlen) .. ").. "[["'
end

-- Create/Extend reversed sorted list of kashida weights:
function Babel.capture_kashida(key, wt)
  wt = tonumber(wt)
  if Babel.kashida_wts then
    for p, q in ipairs(Babel.kashida_wts) do
      if wt == q then
        break
      elseif wt > q then
        table.insert(Babel.kashida_wts, p, wt)
        break
      elseif table.getn(Babel.kashida_wts) == p then
        table.insert(Babel.kashida_wts, wt)
      end
    end
  else
    Babel.kashida_wts = { wt }
  end
  return 'kashida = ' .. wt
end

-- Experimental: applies prehyphenation transforms to a string (letters
-- and spaces).
function Babel.string_prehyphenation(str, locale)
  local n, head, last, res
  head = node.new(8, 0) -- dummy (hack just to start)
  last = head
  for s in string.utfvalues(str) do
    if s == 20 then
      n = node.new(12, 0)
    end
  end
else
  n = node.new(29, 0)
  n.char = s
end
node.set_attribute(n, Babel.attr_locale, locale)
last.next = n
last = n
end
head = Babel.hyphenate_replace(head, 0)
res = ''
for n in node.traverse(head) do
  if n.id == 12 then
    res = res .. ' ' 
  elseif n.id == 29 then
    res = res .. unicode.utf8.char(n.char)
  end
end
tex.print(res)
end

⟨/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 <a>l). 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.

⟨∗basic-r⟩
Babel = Babel or {}
Babel.bidi_enabled = true
local characters = Babel.characters
local ranges = Babel.ranges
local DIR = node.id("dir")
local function dir_mark(head, from, to, outer)
    dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
    local d = node.new(DIR)
    d.dir = '+' .. dir
    node.insert_before(head, from, d)
    d = node.new(DIR)
    d.dir = '-' .. dir
    node.insert_after(head, to, d)
end

function Babel.bidi(head, ispar)
    local first_n, last_n -- first and last char with nums
    local last_es -- an auxiliary 'last' used with nums
    local first_d, last_d -- first and last char in L/R block
    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):
    local strong = ('TRT' == tex.pardir) and 'r' or 'l'
    local strong_lr = (strong == 'l') and 'l' or 'r'
    local outer = strong
    local new_dir = false
    local first_dir = false
    local inmath = false
    local last_lr
    local type_n = ''

    for item in node.traverse(head) do
        -- three cases: glyph, dir, otherwise
        if item.id == node.id('glyph')
            or (item.id == 7 and item.subtype == 2) then
            local itemchar
            if item.id == 7 and item.subtype == 2 then
                itemchar = item.replace.char
            else
                itemchar = item.char
            end
            local chardata = characters[itemchar]
        
        end
end

for nn, et in ipairs(ranges) do
    if not dir then
        for nn, et in ipairs(ranges) do
            if itemchar < et[1] then
                break
            elseif itemchar <= et[2] then
                dir = et[3]
                break
            end
        end
    end
end

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).

```
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 = 'l'
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

148
last_es = item
else if dir == 'cs' then -- it's right - do nothing
else if first_n then -- & if dir = any but en, et, an, es, cs, inc nil
    if strong_lr == 'r' and type_n ~= '' then
        dir_mark(head, first_n, last_n, 'r')
    elseif strong_lr == 'l' and first_d and type_n == 'an' then
        dir_mark(head, first_d, last_d, outer)
        first_d, last_d = nil, nil
    elseif strong_lr == 'l' and type_n ~= '' then
        last_d = last_n
    end
    type_n = ''
end
first_n, last_n = nil, nil
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, whatstis, etc., are ignored:

if dir == 'l' or dir == 'r' then
    if dir ~= outer then
        first_d = first_d or item
        last_d = item
    elseif first_d and dir ~= strong_lr then
        dir_mark(head, first_d, last_d, outer)
        first_d, last_d = nil, nil
    end
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>, respctly, 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.

if dir and not last_lr and dir =~ 'l' and outer =~ 'r' then
    item.char = characters[item.char] and
    characters[item.char].m or item.char
elseif (dir or new_dir) and last_lr ~= item then
    local mir = outer .. strong_lr .. (dir or outer)
    if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
        for ch in node.traverse(node.next(last_lr)) do
            if ch == item then break end
            if ch.id == node.id'glyph' and characters[ch.char] then
                ch.char = characters[ch.char].m or ch.char
            end
        end
    end
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).

if dir == 'l' or dir == 'r' then
    last_lr = item
    strong = dir_real -- Don’t search back - best save now
elseif new_dir then
    last_lr = nil
end
Mirroring the last chars if they are not directed. And make sure any open block is closed, too.

if last_lr and outer =~ 'r' then
    for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
if characters[ch.char] then
  ch.char = characters[ch.char].m or ch.char
end
end

if first_n then
dir_mark(head, first_n, last_n, outer)
end
if first_d then
dir_mark(head, first_d, last_d, outer)
end

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

return node.prev(head) or head
end

⟨/basic-r⟩

And here the Lua code for bidi=basic:

Babel = Babel or {}
-- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
Babel.fontmap = Babel.fontmap or {}
Babel.fontmap[0] = {} -- l
Babel.fontmap[1] = {} -- r
Babel.fontmap[2] = {} -- a/an
Babel.bidi_enabled = true
Babel.mirroring_enabled = true
require('babel-data-bidi.lua')
local characters = Babel.characters
local ranges = Babel.ranges
local DIR = node.id('dir')
local GLYPH = node.id('glyph')
local function insert_implicit(head, state, outer)
  local new_state = state
  if state.sim and state.eim and state.sim ~= state.eim then
    dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
    d = node.new(DIR)
    d.dir = '+' .. dir
    node.insert_before(head, state.sim, d)
    d = node.new(DIR)
    d.dir = '-' .. dir
    node.insert_after(head, state.eim, d)
  end
  new_state.sim, new_state.eim = nil, nil
  return head, new_state
end

local function insert_numeric(head, state)
  local new_state = state
  if state.san and state.ean and state.san ~= state.ean then
    local d = node.new(DIR)
    d.dir = '*TLT'
    _, new = node.insert_before(head, state.san, d)
    if state.san == state.sim then state.sim = new end
    local d = node.new(DIR)
    d.dir = '*' .. dir
    node.insert_after(head, state.eim, d)
  end
  new_state.sim, new_state.eim = nil, nil
  return head, new_state
end

local function insert_implicit(head, state, outer)
  local new_state = state
  if state.sim and state.eim and state.sim ~= state.eim then
    dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
    local d = node.new(DIR)
    d.dir = '+' .. dir
    node.insert_before(head, state.sim, d)
    local d = node.new(DIR)
    d.dir = '-' .. dir
    node.insert_after(head, state.eim, d)
  end
  new_state.sim, new_state.eim = nil, nil
  return head, new_state
end

local function insert_numeric(head, state)
  local new_state = state
  if state.san and state.ean and state.san ~= state.ean then
    local d = node.new(DIR)
    d.dir = 'TLT'
    _, new = node.insert_before(head, state.san, d)
    if state.san == state.sim then state.sim = new end
    local d = node.new(DIR)
  end

150
d.dir = '-TLT'
if state.ean == state.eim then state.eim = new end
new_state.san, new_state.ean = nil, nil
return head, new_state

-- TODO - \hbox with an explicit dir can lead to wrong results
-- <R \hbox dir TLT{<R}>> and <L \hbox dir TRT{<L}>>. A small attempt
-- was made to improve the situation, but the problem is the 3-dir
-- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
-- well.

function Babel.bidi(head, ispar, hdir)
local d -- d is used mainly for computations in a loop
local prev_d = ''
local new_d = false
local nodes = {}
local outer_first = nil
local inmath = false
local glue_d = nil
local glue_i = nil
local has_en = false
local first_et = nil
local has_hyperlink = false
local ATDIR = Babel.attr_dir
local save_outer
local temp = node.get_attribute(head, ATDIR)
if temp then
    temp = temp & 0x3
    save_outer = (temp == 0 and 'l') or
                 (temp == 1 and 'r') or
                 (temp == 2 and 'al')
elseif ispar then -- Or error? Shouldn't happen
    save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
else -- Or error? Shouldn't happen
    save_outer = ('TRT' == hdir) and 'r' or 'l'
end
local outer = save_outer
local last = outer
if save_outer == 'al' then save_outer = 'r' end
local fontmap = Babel.fontmap
for item in node.traverse(head) do
    -- In what follows, #node is the last (previous) node, because the
    -- current one is not added until we start processing the neutrals.
    -- three cases: glyph, dir, otherwise
if item.id == GLYPH
    or (item.id == 7 and item.subtype == 2) then
    local d_font = nil
    local item_r
    if item.id == 7 and item.subtype == 2 then
        item_r = item.replace -- automatic discs have just 1 glyph
    else
        item_r = item
    end
    local chardata = characters[item_r.char]
    d = chardata and chardata.d or nil
    if not d or d == 'nsm' then
        for nn, et in ipairs(ranges) do
            if item_r.char < et[1] then
                break
            elseif item_r.char <= et[2] then
                if not d then d = et[3]
                elseif d == 'nsm' then d_font = et[3]
                end
                break
            end
        end
        d = d or 'l'
    -- A short 'pause' in bidi for mapfont
    d_font = d_font or d
    _d_font = (d_font == 'l' and 0) or
        (d_font == 'nsm' and 0) or
        (d_font == 'r' and 1) or
        (d_font == 'al' and 2) or
        (d_font == 'an' and 2) or nil
    if d_font and fontmap and fontmap[d_font][item_r.font] then
        item_r.font = fontmap[d_font][item_r.font]
    end
    if new_d then
        table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
    if inmath then
        attr_d = 0
    else
        attr_d = node.get_attribute(item, ATDIR)
        attr_d = attr_d & 0x3
    end
    if attr_d == 1 then
        outer_first = 'r'
        last = 'r'
    elseif attr_d == 2 then
        outer_first = 'r'
        last = 'al'
    else
        outer_first = 'l'
        last = 'l'
    end
    outer = last
    has_en = false
    first_et = nil
    new_d = false
    end
    if glue_d then
        if (d == 'l' and 'l' or 'r') ~= glue_d then

        end
    end
table.insert(nodes, {glue_i, 'on', nil})
end

end

else item.id == DIR then
d = nil
if head ~= item then new_d = true end
elseif item.id == node.id'glue' and item.subtype == 13 then
  glue_d = d
  glue_i = item
d = nil
elseif item.id == node.id'math' then
  inmath = (item.subtype == 0)
elseif item.id == 8 and item.subtype == 19 then
  has_hyperlink = true
else
  d = nil
end

-- AL <= EN/ET/ES -- W2 + W3 + W6
if last == 'al' and d == 'en' then
d = 'an' -- W3
elseif last == 'al' and (d == 'et' or d == 'es') then
d = 'on' -- W6
end

-- EN + CS/ES + EN -- W4
if d == 'en' and #nodes >= 2 then
  if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
    and nodes[#nodes-1][2] == 'en' then
    nodes[#nodes][2] = 'en'
  end
end

-- AN + CS + AN -- W4 too, because uax9 mixes both cases
if d == 'an' and #nodes >= 2 then
  if (nodes[#nodes][2] == 'cs')
    and nodes[#nodes-1][2] == 'an' then
    nodes[#nodes][2] = 'an'
  end
end

-- ET/EN -- W5 + W7->l / W6->on
if d == 'et' then
  first_et = first_et or (#nodes + 1)
elseif d == 'en' then
  has_en = true
  first_et = first_et or (#nodes + 1)
elseif first_et then
d may be nil here!
if has_en then
  if last == 'l' then
    temp = 'l' -- W7
  else
    temp = 'en' -- W5
  end
else
end
temp = 'on' -- W6
end
for e = first_et, #nodes do
  if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
end
first_et = nil
has_en = false
end

-- Force mathdir in math if ON (currently works as expected only with 'l')
if inmath and d == 'on' then
d = ('TRT' == tex.mathdir) and 'r' or 'l'
end

if d then
  if d == 'al' then
    d = 'r'
    last = 'al'
  elseif d == 'l' or d == 'r' then
    last = d
  end
  prev_d = d
  table.insert(nodes, {item, d, outer_first})
end
outer_first = nil
end

-- TODO -- repeated here in case EN/ET is the last node. Find a better way of doing things:
if first_et then -- dir may be nil here !
  if has_en then
    if last == 'l' then
      temp = 'l' -- W7
    else
      temp = 'en' -- W5
    end
  else
    temp = 'on' -- W6
  end
  for e = first_et, #nodes do
    if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
  end
end

-- dummy node, to close things
table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})

-------------- neutral ---------------
outer = save_outer
last = outer
local first_on = nil
for q = 1, #nodes do
  local item
  local outer_first = nodes[q][3]
  outer = outer_first or outer
  last = outer_first or last
  if nodes[q][1].id == GLYPH then
    if nodes[q][2] == 'en' then
      temp = 'en' -- W5
    else
      temp = 'on' -- W6
    end
  end
  table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
end

local d = nodes[q][2]
if d == 'an' or d == 'en' then d = 'r' end
if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
if d == 'on' then
  first_on = first_on or q
elseif first_on then
  if last == d then
    temp = d
  else
    temp = outer
  end
  for r = first_on, q - 1 do
    nodes[r][2] = temp
    item = nodes[r][1] -- MIRRORING
    if Babel.mirroring_enabled and item.id == GLYPH
      and temp == 'r' and characters[item.char] then
      local font_mode = ''
      if item.font > 0 and font.fonts[item.font].properties then
        font_mode = font.fonts[item.font].properties.mode
      end
      if font_mode ~= 'harf' and font_mode ~= 'plug' then
        item.char = characters[item.char].m or item.char
      end
      end
      end
      first_on = nil
  end
  if d == 'r' or d == 'l' then last = d end
end
---------- IMPLICIT, REORDER -----------
outer = save_outer
last = outer
local state = {} 
state.has_r = false
for q = 1, #nodes do
  local item = nodes[q][1]
  outer = nodes[q][3] or outer
  local d = nodes[q][2]
  if d == 'nsm' then d = last end ---- W1
  if d == 'en' then d = 'an' end
  local isdir = (d == 'r' or d == 'l')
  if outer == 'l' and d == 'an' then 
    state.san = state.san or item
    state.ean = item
  elseif state.san then
    head, state = insert_numeric(head, state)
  end
  if outer == 'l' then
    if d == 'an' or d == 'r' then -- im -> implicit
      if d == 'r' then state.has_r = true end
  end

state.sim = state.sim or item
state.eim = item
elseif d == 'l' and state.sim and state.has_r then
    head, state = insert_implicit(head, state, outer)
elseif d == 'l' then
    state.sim, state.eim, state.has_r = nil, nil, false
end
else
    if d == 'an' or d == 'l' then
        if nodes[q][3] then -- nil except after an explicit dir
            state.sim = item -- so we move sim 'inside' the group
        else
            state.sim = state.sim or item
        end
    state.eim = item
    elseif d == 'r' and state.sim then
        head, state = insert_implicit(head, state, outer)
    elseif d == 'r' then
        state.sim, state.eim = nil, nil
    end
end
else
    if isdir then
        last = d -- Don't search back - best save now
    elseif d == 'on' and state.san then
        state.san = state.san or item
        state.ean = item
    end
end
head = node.prev(head) or head

-------------- FIX HYPERLINKS ----------------
if has_hyperlink then
    local flag, linking = 0, 0
    for item in node.traverse(head) do
        if item.id == DIR then
            if item.dir == '+TRT' or item.dir == '+TLT' then
                flag = flag + 1
            elseif item.dir == '-TRT' or item.dir == '-TLT' then
                flag = flag - 1
            end
        elseif item.id == 8 and item.subtype == 19 then
            linking = flag
        elseif item.id == 8 and item.subtype == 20 then
            if linking > 0 then
                if item.prev.id == DIR and
                    (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
                    d = node.new(DIR)
                    d.dir = item.prev.dir
                    node.remove(head, item.prev)
                    node.insert_after(head, item, d)
                end
            linking = 0
        end
        end
    end
    return head
end
11 Data for CJK

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

```plaintext
[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.

```
\ProvideLanguage{nil}[[date] v[version]] Nil language
\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.

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

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

```
\providehyphenmins[\CurrentOption]{m@ne\m@ne}
```

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

```
\captionnil
\datenil
```

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

```
\def\bbl@inidata{nil}{
  \bbl@elt{identification}\{tag.ini\}\{und\}%
  \bbl@elt{identification}\{load.level\}\{0\}%
  \bbl@elt{identification}\{charset\}\{utf8\}%
  \bbl@elt{identification}\{version\}\{1.0\}%
  \bbl@elt{identification}\{date\}\{2022-05-16\}%
  \bbl@elt{identification}\{name.local\}\{nil\}%
  \bbl@elt{identification}\{name.english\}\{nil\}%
  \bbl@elt{identification}\{name.babel\}\{nil\}%
  \bbl@elt{identification}\{tag.bcp47\}\{und\}%
  \bbl@elt{identification}\{language.tag.bcp47\}\{und\}%
  \bbl@elt{identification}\{tag.opentype\}\{dflt\}%
  \bbl@elt{identification}\{script.name\}\{Latin\}%
  \bbl@elt{identification}\{script.tag.bcp47\}\{Latin\}%
  \bbl@elt{identification}\{script.tag.opentype\}\{DFLT\}%
```
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.

13.1 Islamic

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

```
\ExplSyntaxOn
\begin{Compute Julian day}
% == islamic (default)
% Not yet implemented
\def\bbl@ca@islamic-civil-{-1}
\def\bbl@ca@islamic-civil-{-2}
\def\bbl@ca@islamic-vl@x#1#2-#3-#4\@@#5#6#7{
  \edef\tempa{\fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}
  \edef\tempb{\fp_eval:n{ floor(((30*(\tempa-1948439.5)) + 10646)/10631) }}

\ExplSyntaxOff
```

The Civil calendar.

```
\def\bbl@cs@isltojd#1#2#3{
  ((#3 + ceil(29.5 * (#2 - 1)) +
  (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
  1948439.5 - 1) }
\def\bbl@cs@isltojd#1#2#3{
  ((#2 <= 2) ? 0 : ((#1-1)/100) + floor((#1-1)/400) +
  floor((#1-1)/30) +
  floor((#1-1)/4) +
  floor((30*(#2-2)/12) +
  (365 * (#1 - 1)) +
  1948439.5 - 1) }
```

The 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`.

13.1 Islamic

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

```
\def\bbl@cfm#1#2{\@floor{#1/#2}}
\def\bbl@cs@gregleap#1{\bbl@fpmod{#1}{4} == 0} &&
\def\bbl@cs@jd#1#2#3{\fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
  floor((#1 - 1) / 4) +
  floor((#1 - 1) / 100) +
  floor((#1 - 1) / 400) +
  floor((367 * #2 - 362) / 12) +
  ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3 )}}
```

13.1 Islamic

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

```
\def\bbl@cfm#1#2{\@floor{#1/#2}}
\def\bbl@cs@gregleap#1{\bbl@fpmod{#1}{4} == 0} &&
\def\bbl@cs@jd#1#2#3{\fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
  floor((#1 - 1) / 4) +
  floor((#1 - 1) / 100) +
  floor((#1 - 1) / 400) +
  floor((367 * #2 - 362) / 12) +
  ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3 )}}
```

The 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`.

13.1 Islamic

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

```
\def\bbl@cfm#1#2{\@floor{#1/#2}}
\def\bbl@cs@gregleap#1{\bbl@fpmod{#1}{4} == 0} &&
\def\bbl@cs@jd#1#2#3{\fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
  floor((#1 - 1) / 4) +
  floor((#1 - 1) / 100) +
  floor((#1 - 1) / 400) +
  floor((367 * #2 - 362) / 12) +
  ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3 )}}
```

The calendars.
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).

\def\bbl@cs@umalqura@data{56660, 56690, 56719, 56749, 56778, 56808, 56837, 56867, 56897, 56926, 56956, 56985, 57015, 57044, 57074, 57103, 57133, 57162, 57192, 57221, 57250, 57280, 57310, 57340, 57369, 57399, 57429, 57458, 57487, 57517, 57546, 57576, 57605, 57634, 57664, 57694, 57723, 57753, 57783, 57813, 57842, 57871, 57901, 57930, 57959, 57989, 58018, 58048, 58077, 58107, 58137, 58167, 58196, 58226, 58255, 58285, 58314, 58343, 58373, 58402, 58432, 58461, 58491, 58521, 58551, 58580, 58610, 58639, 58669, 58698, 58727, 58757, 58786, 58816, 58845, 58875, 58905, 58934, 58964, 58994, 59023, 59053, 59082, 59111, 59141, 59170, 59200, 59229, 59259, 59288, 59318, 59348, 59377, 59407, 59436, 59466, 59495, 59525, 59554, 59584, 59613, 59643, 59672, 59702, 59731, 59761, 59791, 59820, 59850, 59897, 59926, 59959, 59997, 59999, 60027, 60056, 60086, 60115, 60145, 60174, 60204, 60234, 60264, 60293, 60323, 60352, 60381, 60411, 60440, 60469, 60499, 60528, 60558, 60588, 60618, 60648, 60677, 60707, 60736, 60765, 60795, 60824, 60853, 60883, 60912, 60942, 60972, 60900, 60909, 61061, 61110, 61129, 61179, 61208, 61237, 61267, 61296, 61326, 61356, 61385, 61415, 61445, 61474, 61504, 61533, 61563, 61592, 61621, 61651, 61680, 61710, 61739, 61769, 61799, 61828, 61858, 61888, 61917, 61947, 61976, 62006, 62035, 62064, 62094, 62123, 62153, 62182, 62212, 62242, 62271, 62301, 62331, 62360, 62390, 62419, 62448, 62478, 62507, 62537, 62566, 62596, 62625, 62655, 62685, 62715, 62744, 62774, 62803, 62832, 62862, 62891, 62921, 62950, 62980, 63009, 63039, 63069, 63099, 63128, 63157, 63187, 63216, 63246, 63275, 63305, 63334, 63363, 63393, 63423, 63453, 63482, 63512, 63541, 63571, 63600, 63630, 63659, 63689, 63718, 63747, 63777, 63807, 63836, 63866, 63895, 63925, 63955, 63984, 64014, 64043, 64073, 64102, 64131, 64161, 64190, 64220, 64249, 64279, 64309, 64339, 64368, 64398, 64427, 64457, 64486, 64515, 64545, 64574, 64603, 64633, 64663, 64692, 64722, 64752, 64782, 64811, 64841, 64870, 64899, 64929, 64958, 64987, 65017, 65047, 65076, 65106, 65136, 65166, 65195, 65225, 65254, 65283, 65313, 65342, 65371, 65401, 65431, 65460, 65490, 65520}

\ExplSyntaxOff

\bbl@add\bbl@precalendar{%
13.2 Hebrew

This is basically the set of macros written by Michail Rozman in 1991, with corrections and adaptions 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
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@hebrelapsedmonths#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}\
8173 \tmpa=years\19-years this cycle
8174 \tmpc=\tmpb
8175 \multiply \tmpb by 12
8176 \advance \#2 by \tmpb
8177 \multiply \tmpc by 7
8178 \divide \tmpc by 19
8179 \advance \#2 by \tmpc
8180 \global\bbl@cntcommon=#2\relax
8181 \#2=\bbl@cntcommon}
8182 \def\bbl@hebrelapseddays#1#2{%
8183 {\countdef\tmpa=0
8184 \countdef\tmpb=1
8185 \countdef\tmpc=2
8186 \bbl@hebrelapsedmonths{#1}{#2}\
8187 \tmpa=#2\relax
8188 \multiply \tmpa by 13753
8189 \advance \tmpa by 5604
8190 \bbl@remainder{\tmpa}{25920}{\tmpc}\
8191 \tmpc = ConjunctionParts
8192 \divide \tmpa by 25920
8193 \tmpc=\tmpb
8194 \advance \tmpc by \tmpb
8195 \global\bbl@cntcommon=#2\relax
8196 \#2=\bbl@cntcommon
161
\multiply \rom{#2} by \num{29}
\advance \rom{#2} by \rom{1}
\advance \rom{#2} by \rom{\tmpa}
\@bblremainder{\rom{#2}}{7}{\rom{\tmpa}}%
\ifnum \rom{\tmpc} < 19440
\ifnum \rom{\tmpc} < 9924
\else
\@bblcheckleaphebryear{\rom{#1}}% of a common year
\if@bblhebrleap
\else
\advance \rom{#2} by \rom{1}
\fi
\fi
\ifnum \rom{\tmpc} < 16789
\else
\ifnum \rom{\tmpa} = \rom{2}
\@bblcheckleaphebryear{\rom{#1}}% at the end of leap year
\if@bblhebrleap
\advance \rom{#2} by \rom{1}
\fi
\fi
\else
\advance \rom{#2} by \rom{1}
\fi
\@bblremainder{\rom{#2}}{7}{\rom{\tmpa}}%
\if\rom{\tmpa} = \rom{0}
\advance \rom{#2} by \rom{1}
\else
\if\rom{\tmpa} = \rom{3}
\advance \rom{#2} by \rom{1}
\else
\if\rom{\tmpa} = \rom{5}
\advance \rom{#2} by \rom{1}
\fi
\fi
\fi
\global\@bblcntcommon=\rom{#2}\relax%
\def\@bbl@daysinhebryear#1#2{%\countdef\tmpe=12
\@bbl@hebreldayspriormonths#1#2#3{%\countdef\tmpf=14
\ifcase \rom{#1} \relax
0 \or
10 \or
30 \or
50 \or
80 \or
110 \or
140 \or
148 \or
177 \or
207 \or
\fi
\global@addto@macro\@bblcntcommon=\rom{\tmpf}}
\#236 \or
\#266 \or
\#295 \or
\#325 \or
\#400
\fi
\ifbbl@checkleaphebryear\#2%
  \ifnum \#1 > 6
    \advance \#3 by 30
  \fi
\fi
\fi
\ifbbl@daysinhebryear\#2\{\tmpf\}%
\ifnum \#1 > 3
  \ifnum \tmpf=353
    \advance \#3 by -1
  \fi
  \ifnum \tmpf=383
    \advance \#3 by -1
  \fi
  \ifnum \#1 > 2
    \ifnum \tmpf=355
      \advance \#3 by 1
    \else
      \ifnum \tmpf=385
        \advance \#3 by 1
      \fi
    \fi
  \fi
  \global\bbl@cntcommon=\#3\relax}
\#3=\bbl@cntcommon
\def\bbl@hebrfromgreg#1#2#3#4#5#6{%
  \countdef\tmpx= 17
  \countdef\tmpy= 18
  \countdef\tmpz= 19
  \#6=\#3\relax
  \global\advance \#6 by 3761
  \bbl@absfromhebr\#1\#2\#3\#4\% 
  \#5=\#4\relax
  \divide \#5 by 30
  \loop
    \bbl@hebrdayspriormonths\#5\{\#6\}\{\tmpx\}% 
    \ifnum \tmpx < \#4\relax
      \advance \#5 by 1
      \tmpy=\tmpx
    \else
      \fi
  \repeat
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 LPLP 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).

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.
13.5 Buddhist

That's very simple.

\begin{macrocode}
\ExplSyntaxOn
\ComputeJulianDay
\def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{
\edef#4{\number\numexpr#1+543\relax}
\edef#5{#2}
\edef#6{#3}
\ExplSyntaxOff
\subsection{Chinese}

Brute force, with the Julian day of first day of each month. The table has been computed with the help of \texttt{python-lunardate} by Ricky Yeung, GPLv2 (but the code itself has not been used). The range is 2015-2044.

\begin{macrocode}
\ExplSyntaxOn
\ComputeJulianDay
\def\bbl@ca@chinese#1-#2-#3\@@#4#5#6{
\edef\bbl@tempd{\fp_eval:n{\bbl@cs@jd{#1}{#2}{#3} - 2457072.5}}
\count@\z@
\@tempcnta=2015
\bbl@foreach\bbl@cs@chinese@data{\ifnum##1>\bbl@tempd\else\advance\count@\@ne\ifnum\count@>12\count@\@ne\advance\@tempcnta\@ne\fi\bbl@xin{{,##1,},{,\bbl@cs@chinese@leap,}}\ifin@\advance\count@\m@ne\edef\bbl@tempe{\the\numexpr\count@+12\relax}\else\edef\bbl@tempe{\the\count@}\fi\edef\bbl@tempb{##1}\fi} \count@\z@
\@tempcnta=2015
\bbl@foreach\bbl@cs@chinese@data{\ifnum##1>\bbl@tempd\else\advance\count@\@ne\ifnum\count@>12\count@\@ne\edef\bbl@xing{##1},\bbl@cs@chinese@leap,}\ifin@\advance\count@\m@ne\edef\bbl@tempe{\the\numexpr\count@+12\relax}\else\edef\bbl@tempe{\the\count@}\fi\edef\bbl@tempb{##1}\fi}
14 Support for Plain \TeX\ (plain.def)

14.1 Not renaming hyphen\texttt{.tex}

As Don Knuth has declared that the filename \texttt{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 \TeX{}-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 locally\texttt{hyphen.tex} or whatever they like, but they mustn't diddle with \texttt{hyphen.tex} (or \texttt{plain.tex} except to preload additional fonts).

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

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

\texttt{\catcode\{=1 \catcode\}=2}

\ExplSyntaxOff

\endinput

\def\bbl@cs@chinese@leap{\edef#4{\the\@tempcnta}}
\edef#5{\bbl@tempe}
\edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
\def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325, 354,384,413,443,472,501,531,560,590,620,649,679,709,739, 768,797,827,856,885,915,944,974,1003,1033,1063,1093,1123, 1152,1182,1212,1242,1272,1302,1332,1362,1392,1422,1452,1482, 1512,1542,1572,1602,1632,1662,1692,1722,1752,1782,1812,1842, 1872,1902,1932,1962,1992,2022,2052,2082,2112,2142,2172,2202, 2232,2262,2292,2322,2352,2382,2412,2442,2472,2502,2532,2562, 2592,2622,2652,2682,2712,2742,2772,2802,2832,2862,2892,2922, 2952,2982,3012,3042,3072,3102,3132,3162,3192,3222,3252,3282, 3312,3342,3372,3402,3432,3462,3492,3522,3552,3582,3612,3642, 3672,3702,3732,3762,3792,3822,3852,3882,3912,3942,3972,3992, 4012,4042,4072,4102,4132,4162,4192,4222,4252,4282,4312,4342, 4372,4402,4432,4462,4492,4522,4552,4582,4612,4642,4672,4702, 4732,4762,4792,4822,4852,4882,4912,4942,4972,5002,5032,5062, 5092,5122,5152,5182,5212,5242,5272,5302,5332,5362,5392,5422, 5452,5482,5512,5542,5572,5602,5632,5662,5692,5722,5752,5782, 5812,5842,5872,5902,5932,5962,5992,6022,6052,6082,6112,6142, 6172,6202,6232,6262,6292,6322,6352,6382,6412,6442,6472,6492, 6522,6552,6582,6612,6642,6672,6702,6732,6762,6792,6822,6852, 6882,6912,6942,6972,6992,7022,7052,7082,7112,7142,7172,7202, 7232,7262,7292,7322,7352,7382,7412,7442,7472,7502,7532,7562, 7592,7622,7652,7682,7712,7742,7772,7802,7832,7862,7892,7922, 7952,7982,8012,8042,8072,8102,8132,8162,8192,8222,8252,8282, 8312,8342,8372,8402,8432,8462,8492,8522,8552,8582,8612,8642, 8672,8702,8732,8762,8792,8822,8852,8882,8912,8942,8972,9002, 9032,9062,9092,9122,9152,9182,9212,9242,9272,9302,9332,9362, 9392,9422,9452,9482,9512,9542,9572,9602,9632,9662,9692,9722, 9752,9782,9812,9842,9872,9902,9932,9962,9992,10022,10052,10082, 10112,10142,10172,10202,10232,10262,10292,10322,10352,10382, 10412,10442,10472,10502,10532,10562,10592,10622,10652,10682, 10712,10742,10772,10802,10832,10862,10892,10922,10952,10982, 11012,11042,11072,11102,11132}
If a file called hyphen.cfg can be found, we make sure that \input 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.

14.2 Emulating some \LaTeX features

The file babel.def expects some definitions made in the \LaTeX2ε 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 and 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).

14.3 General tools

A number of \LaTeX macro's that are needed later on.
LaTeX has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```latex
\def\@onlypreamble#1{\expandafter\gdef\expandafter\@preamblecmds\expandafter{\@preamblecmds\do#1}}
```

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

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

\def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
\@onlypreamble\AtEndOfPackage
\def\@endofldf{}
\@onlypreamble\@endofldf
\let\bbl@afterlang\@empty
\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.

\catcode`\&=\z@
\ifx&if@filesw\@undefined
\expandafter\let\csname if@filesw\expandafter\endcsname\csname iffalse\endcsname
\fi
\catcode`\&=4

Mimic \LaTeX's commands to define control sequences.

\def\newcommand{\@star@or@long\new@command}
\def\new@command#1{\@testopt{\@newcommand#1}0}
\def\@newcommand#1[#2]{\@ifnextchar\[\@xargdef#1[#2]{}\@argdef#1[#2]}
\long\def\@argdef#1[#2]#3{\@yargdef#1@ne{#2}{#3}}
\long\def\@xargdef#1[#2][#3]#4{\expandafter\def\expandafter#1\expandafter{\@protected@testopt#1\csname\string#1\endcsname{#3}}\expandafter\@yargdef\csname\string#1\endcsname\tw@{#2}{#4}}
\def\providecommand{\@star@or@long\provide@command}
\def\provide@command#1{\begingroup\escapechar\m@ne\xdef\@gtempa{{\string#1}}\endgroup\expandafter\@ifundefined\@gtempa1\fi}
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.

\def\bbl@tempa\{\csname newif\endcsname\&ifin@\}
\catcode`\&=4
\ifx\in@\@undefined
\def\in@#1#2{\def\in@@##1#1##2##3\in@@{\ifx\in@##2\in@false\else\in@true\fi}\in@@#2#1\in@\in@@}
\else
\let\bbl@tempa\@empty
\fi
\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).

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

The \LaTeX{} macro \if@if@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.

\def\@if@aded#1#2#3#4{\relax}

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

\ifx\tempcnta\@undefined
\csname newcount\endcsname\tempcnta\relax
\fi
\ifx\tempcntb\@undefined
\csname newcount\endcsname\tempcntb\relax
\fi
To prevent wasting two counters in \LaTeX{} (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (\texttt{\count10}).

\verbatiminput{page-1}

\subsection{Encoding related macros}

Code from \texttt{ltoutenc.dtx}, adapted for use in the plain \TeX{} environment.
\fi
\def\@changed@cmd#1\string#2{%  
\ifx\protect\@typeset@protect  
\expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax  
\expandafter\ifx\csname ?\string#1\endcsname\relax  
\expandafter\def\csname ?\string#1\endcsname{%  
\@changed@x@err{#1}%  
}%  
\fi  
\global\expandafter\let\csname\cf@encoding \string#1\expandafter\endcsname\csname ?\string#1\endcsname\fi  
\csname\cf@encoding\string#1\endcsname  
\else  
\noexpand#1%  
\fi  
}\def\@changed@x@err#1{%  
\errhelp{Your command will be ignored, type <return> to proceed}%  
\errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
\def\DeclareTextCommandDefault#1{%  
\DeclareTextCommand#1?%}
\def\ProvideTextCommandDefault#1{%  
\ProvideTextCommand#1?%}
\expandafter\let\csname OT1-cmd\endcsname\@current@cmd\expandafter\let\csname?-cmd\endcsname\@changed@cmd
\def\DeclareTextAccent#1#2#3{%  
\DeclareTextCommand#1{#2}\[1\]{\accent#3 ##1}
\def\DeclareTextCompositeCommand#1#2#3#4{%  
\expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname\ifx\reserved@b\reserved@c\expandafter\expandafter\expandafter\ifx\@car\reserved@a\relax\relax\@nil\@text@composite\else\edef\reserved@b##1{%  
\def\expandafter\noexpand\csname#2\string#1\endcsname####1{%  
\noexpand\@text@composite\noexpand\csname#2\string#1\endcsname####1
\noexpand\@empty
\noexpand\@text@composite
{##1}\
\expandafter\reserved@b\expandafter{\reserved@a{##1}}\fi  
\expandafter\def\csname\expandafter\string\csname#2\endcsname\string#1-\string#3\endcsname{#4}\else\errhelp{Your command will be ignored, type <return> to proceed}%  
\errmessage{\string\DeclareTextCompositeCommand\space used on inappropriate command \protect#1}\fi  
\fi  
\def\expandafter\@strip@args\meaning\reserved@a{-\@strip@args}
Currently we only use the \HKTExt2c method for accents for those that are known to be made active in some language definition file.

The following control sequences are used in babel.def but are not defined for \LaTeXX.

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

And a few more “dummy” definitions.

\let\bbl@opt@shorthands@nil
\let\bbl@ifshorthand#1#2#3{#2}
\let\bbl@language@opts@empty
\let\bbl@ensureinfo@gobble
\let\bbl@provide@locale@relax
\ifx\babeloptionstrings@undefined
  \let\scriptsize\sevenrm
\fi
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