Babel

Localization and internationalization

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

1 Identification and loading of required files

*Code documentation is still under revision.*

The babel package after unpacking consists of the following files:

- **babel.sty** is the \LaTeX{} package, which set options and load language styles.
- **babel.def** is loaded by Plain.
- **switch.def** defines macros to set and switch languages (it loads part babel.def).
- **plain.def** is not used, and just loads babel.def, for compatibility.
- **hyphen.cfg** is the file to be used when generating the formats to load hyphenation patterns.

There some additional \tex, \def and \lua files

The babel installer extends docstrip with a few “pseudo-guards” to set “variables” used at installation time. They are used with \texttt{\langle@name@\rangle} at the appropriate places in the source code and defined with either \texttt{\langle\langle name=value \rangle\rangle}, or with a series of lines between \texttt{\langle@name@\rangle} and \texttt{\langle@/name@\rangle}. The latter is cumulative (eg, with \texttt{More package options}). That brings a little bit of literate programming. The guards \texttt{-<name>} and \texttt{<+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 dtex. 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

\langle\langle version=24.2\rangle\rangle
\langle\langle date=2024/02/07\rangle\rangle

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 \texttt{\bbl@afterfi}, will not change.

We define some basic macros which just make the code cleaner. \texttt{\bbl@add} is now used internally instead of \texttt{\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.

\langle\langle Basic macros\rangle\rangle ≡
\langle\langle *Basic macros\rangle\rangle
\langle\langle \bbl@trace{Basic macros}\rangle\rangle
\langle\langle \def\bbl@stripslash{\expandafter\@gobble\string}\rangle\rangle
\langle\langle \def\bbl@add#1#2{\bbl@ifunset{\bbl@stripslash#1}{{\def#1{#2}}}{}{\expandafter\def\expandafter#1\expandafter{#1#2}}}\rangle\rangle
\langle\langle \def\bbl@xin@{\@expandtwoargs\in@}\rangle\rangle
\langle\langle \def\bbl@carg#1#2{\expandafter#1\csname#2\endcsname}\rangle\rangle
\langle\langle \def\bbl@ncarg#1#2#3{\expandafter#1\expandafter#2\csname#3\endcsname}\rangle\rangle
\langle\langle \def\bbl@ccarg#1#2#3{\expandafter#1\csname#2\expandafter\endcsname\csname#3\endcsname}\rangle\rangle
\langle\langle \def\bbl@csarg#1#2{\expandafter#1\csname bbl@#2\endcsname}\rangle\rangle
\langle\langle \def\bbl@cs#1{\csname bbl@#1\endcsname}\rangle\rangle
\langle\langle \def\bbl@cl#1{\csname bbl@#1@\languagename\endcsname}\rangle\rangle

3
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.

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. These macros will break if another \if...\fi statement appears in one of the arguments and it is not enclosed in braces.

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

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.

To check if a macro is defined, we create a new macro, which does the same as \@ifundefined. However, in an \epsilon-tex engine, it is based on \ifcsname, which is more efficient, and does not waste
memory. Defined inside a group, to avoid `\ifcsname ... \relax` by the `\csname ... test.`

\begin{group}
\gdef\bbl@ifunset#1{%\expandafter\ifx\csname#1\endcsname\relax
\else \expandafter\@firstoftwo \fi}
\bbl@ifunset{ifcsname}{\bbl@ifunset{ifcsname}{\ifcsname#1\endcsname\expandafter\ifx\csname#1\endcsname\relax
\bbl@afterelse\expandafter\@firstoftwo \else \bbl@afterfi\expandafter\@secondoftwo \fi}}
\end{group}

\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.

\def\bbl@ifblank#1{%\bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
\long\def\bbl@ifblank@i#1#2\@nil#3#4\@nil{#4}
\def\bbl@ifset#1#2#3{%\bbl@ifunset{#1}{#3}{\bbl@exp{\bbl@ifblank{\@nameuse{#1}}}{#3}{#2}}}

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

\def\bbl@forkv#1#2{%\def\bbl@forcmd##1{#2}\bbl@fornext#1,\@nil,}
\def\bbl@fornext#1,{%\ifx\@nil#1\relax\else \bbl@ifblank{#1}{}{\bbl@forkv@eq#1=#2=#3\@nil#4}{\expandafter\bbl@fornext}\fi}
\def\bbl@forkv@eq#1=#2=#3\@nil#4{%\bbl@trim@def\bbl@forkv@a{#1}{\expandafter\bbl@kvcmd\expandafter{\bbl@forkv@a}}{#2}{#4}}

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

\def\bbl@foreach#1{%\expandafter\bbl@vforeach\expandafter{#1}}
\bbl@replace Returns implicitly `\toks@` with the modified string.

\def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3 \toks@{} \def\bbl@replace@aux#1#2#3#4%
An extension to the previous macro. It takes into account the parameters, and it is string based (ie, if you replace \relax by ho, then \relax becomes \rho). No checking is done at all, because it is not a general purpose macro, and it is used by babel only when it works (an example where it does not work is in \bbl@TG@@date, and also fails if there are macros with spaces, because they are retokenized). It may change! (or even merged with \bbl@replace; I'm not sure checking the replacement is really necessary or just paranoia).

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

```latex
\def\bbl@bsphack{% 
  \ifhmode 
    \hskip\z@skip 
    \def\bbl@esphack{\loop\ifdim\lastskip>\z@ \unskip \repeat \unskip} \else 
    \let\bbl@esphack\@empty 
  \fi}
```

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

```latex
\def\bbl@cased{% 
  \ifx\oe\OE 
    \expandafter\in@ \expandafter{\expandafter\OE \expandafter}{\oe} \ifin@ 
      \bbl@afterelse \expandafter\MakeUppercase 
    \else 
      \bbl@afterfi \expandafter\MakeLowercase 
    \fi 
  \else 
    \expandafter\@firstofone 
  \fi}
```

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

```latex
\def\bbl@extras@wrap#1#2#3{% 1:in-test, 2:before, 3:after 
  \toks@ \expandafter\expandafter\expandafter{\csname extras\languagename\endcsname} \bbl@exp{\in@{#1}{\the\toks@}} \ifin@ \else 
    \@temptokena{#2} \edef\bbl@tempc{\the\@temptokena\the\toks@} \toks@ \expandafter{\bbl@tempc#3} \expandafter\edef\csname extras\languagename\endcsname{\the\toks@} \fi \fi}
```

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

```latex
\def\bbl@extras@wrap1#2#3{% 1:in-test, 2:before, 3:after 
  \toks@ \expandafter\expandafter\expandafter{\csname extras\languagename\endcsname} \bbl@exp{\in@{#1}{\the\toks@}} \ifin@ \else 
    \@temptokena{#2} \edef\bbl@tempc{\the\@temptokena\the\toks@} \toks@ \expandafter{\bbl@tempc#3} \expandafter\edef\csname extras\languagename\endcsname{\the\toks@} \fi}
```

### 3.1 Multiple languages

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

```latex
\def\bbl@extras@wrap1#2#3{% 1:in-test, 2:before, 3:after 
  \toks@ \expandafter\expandafter\expandafter{\csname extras\languagename\endcsname} \bbl@exp{\in@{#1}{\the\toks@}} \ifin@ \else 
    \@temptokena{#2} \edef\bbl@tempc{\the\@temptokena\the\toks@} \toks@ \expandafter{\bbl@tempc#3} \expandafter\edef\csname extras\languagename\endcsname{\the\toks@} \fi}
```
Another counter is used to keep track of the allocated languages. \TeX{} and \LaTeX{} reserves for this purpose the count 19.

\addlanguage

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

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.

```latex
\@ifpackagewith{babel}{debug}{{\providecommand\bbl@trace[1]{\message{\[ #1 \]}}}%\let\bbl@debug\@firstofone\ifx\directlua\@undefined\else\directlua{ Babel = Babel or {}\Babel.debug = true }%\input{babel-debug.tex}%\fi}{\providecommand\bbl@trace[1]{}%\let\bbl@debug\@gobble\ifx\directlua\@undefined\else\directlua{ Babel = Babel or {}\Babel.debug = false }%\fi}{\def\bbl@error#1{% Implicit #2#3#4\begingroup\catcode`\=0 \catcode`\=12 \catcode`\=12\input errbabel.def\endgroup\bbl@error{#1}}}{\def\bbl@warning#1{%\begingroup\def\MessageBreak{\PackageWarning{babel}{#1}}\endgroup\bbl@warning{#1}}}{\def\bbl@infowarn#1{%\begingroup\def\MessageBreak{\PackageNote{babel}{#1}}\endgroup\bbl@infowarn{#1}}}{\def\bbl@info#1{%\begingroup\def\MessageBreak{\PackageInfo{babel}{#1}}\begingroup\def\MessageBreak{\PackageNote{babel}{#1}}\endgroup\bbl@info{#1}}%\ifx\language\@undefined\csname newcount\endcsname\language\fi}%\countdef\last@language=19\def\addlanguage{%\csname newlanguage\endcsname}%\endgroup
```

This file also takes care of a number of compatibility issues with other packages an 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.

\begin{Basic macros}
@ifpackagewith{babel}{silent}
{\let\bbl@info\@gobble
 \let\bbl@infowarn\@gobble
 \let\bbl@warning\@gobble}
\endgroup
%\def\AfterBabelLanguage#1{\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.
\ifx\bbl@languages\@undefined\else
\begingroup
\catcode`^^I=12
@ifpackagewith{babel}{showlanguages}{
\begin{group}
\def\bbl@elt#1#2#3#4{\wlog{#2\texttt{\#1}\texttt{\#3}\texttt{\#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%

\begin{Basic macros}
@ifpackagewith{babel}{base}{
\let\bbl@onlyswitch\@empty
\let\bbl@provide@locale\relax
\input babel.def
\let\bbl@onlyswitch\@undefined
\ifx\directlua\@undefined
\DeclareOption*{\bbl@patterns{\CurrentOption}}%
\else
\input luababel.def
\DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
\fi
\DeclareOption{base}{}
\DeclareOption{showlanguages}{}
\ProcessOptions
\global\expandafter\let\csname opt@babel.sty\endcsname\relax
\global\expandafter\let\csname ver@babel.sty\endcsname\relax
\global\let\@ifl@ter@@\@ifl@ter
\def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%

3.3 base

The first 'real' option to be processed is base, which set the hyphenation patterns then resets\texttt{ver@babel.sty} so that \texttt{epstopdf} forgets about the first loading. After a subset of babel.def has been loaded (the old switch.def) and \texttt{\AfterBabelLanguage} defined, it exits. Now the base option. With it we can define (and load, with \texttt{luatex}) hyphenation patterns, even if we are not interested in the rest of \texttt{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 \BabelModifiers at \bbl@load@language; when no modifiers have been given, the former is \relax. How modifiers are handled are left to language styles; they can use \in@, loop them with \@for or load keyval, for example.

\bbl@trace{key=value and another general options}
\bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
\def\bbl@tempb#1.#2{% Remove trailing dot
#1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi%
}\def\bbl@tempe#1=#2\@@{% \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}
}\def\bbl@tempd#1.#2\@nnil{% TODO. Refactor lists?
\ifx\@empty#2%
\edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
\else
\in@{$modifiers$}{$#1$}% TODO. Allow spaces.
\in@
\edef\bbl@tempc{
\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
\else
\in@{=}{#1}%
\in@
\edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
\bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
\fi
\fi
\fi
\fi
\else
\in@{,provide=}[,!1]%
\ifin@
\edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
\else
\in@{+}{,!1}%
\in@
\edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
\bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
\fi
\fi
\fi
\fi
\bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
\expandafter\let\csname opt@babel.sty\endcsname\bbl@tempc

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

\DeclareOption{KeepShorthandsActive}{}
\DeclareOption{activeacute}{}
\DeclareOption{activegrave}{}
\DeclareOption{debug}{}
\DeclareOption{noconfigs}{}
\DeclareOption{showlanguages}{}
\DeclareOption{silent}{}
% \DeclareOption{mono}{}
\DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
\chardef\bbl@iniflag\z@
\DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main -> +1
\DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@} % add = 2
\DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % add + main
% A separate option
\newif\ifbbl@single
\DeclareOption{selectors=off}{\bbl@singletrue}
\ifbbl@single
\chardef\bbl@autoload@options\@empty
\fi
\bbl@foreach\bbl@tempa{\bbl@tempc\empty}
\bbl@foreach\csarg\bbl@tempa{\bbl@tempd\empty}
\expandafter\let\csname opt@babel.sty\endcsname\bbl@tempc\bbl@tempa
Handling of package options is done in three passes. (I [JBL] am not very happy with the idea, anyway.) The first one processes options which has been declared above or follow the syntax \texttt{<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.

\begin{verbatim}
\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
\end{verbatim}

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

\begin{verbatim}
\def\bbl@tempa#1=#2\bbl@tempa{\%
   \bbl@csarg\ifx{opt@#1}\@nnil
   \bbl@csarg\edef{opt@#1}{#2}%
   \else
   \bbl@error{bad-package-option}{#1}{#2}{}%
\fi}
\end{verbatim}

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

\begin{verbatim}
\let\bbl@language@opts\@empty
\DeclareOption*{\%
   \bbl@xin@\string=\CurrentOption\%
   \ifin@\expandafter\bbl@tempa\CurrentOption\bbl@tempa\%
   \else\bbl@add@list\bbl@language@opts{\CurrentOption}\%
   \fi}
\end{verbatim}

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

\begin{verbatim}
\ProcessOptions*
\ifx\bbl@opt@provide\@nnil\%
   \let\bbl@opt@provide\@empty % %%% MOVE above
\else\%
   \chardef\bbl@iniflag@one
   \bbl@exp\string/**/*.\bbl@forkv\string\@nameuse{\@raw@opt@babel.sty}\{}{\%
   \in@{,provide,}{,#1,}\
   \ifin@\%
   \def\bbl@opt@provide{#2}\
   \bbl@replace\bbl@opt@provide{;}{,}\
   \fi\
   \bbl@replace\bbl@opt@provide{}{,}\
\fi\%
\end{verbatim}

\subsection*{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 \texttt{\bbl@ifshorthand} is always true, and it is always false if shorthands is empty. Also, some code makes sense only with shorthands=...
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}{}
\fi

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

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

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.

\ifx\bbl@opt@safe\@undefined\def\bbl@opt@safe{BR}\fi

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]{#3}\newcommand\IfBabelLayout[1]{\@expandtwoargs\selectlanguage@x{\bbl@opt@headfoot}}

For package ⟨∗ core⟩
3.6 Interlude for Plain

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

```
\ifx\ldf@quit\@undefined\else
\endinput\fi % Same line!
\ProvidesFile{babel.def}\[⟨⟨date⟩⟩v⟨⟨version⟩⟩Babel common definitions]
\ifx\AtBeginDocument\@undefined % TODO. change test.
⟨⟨Emulate LaTeX⟩⟩
\fi
⟨⟨Basic macros⟩⟩
```

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

4 Multiple languages

This is not a separate file (switch.def) anymore. Plain \TeX version 3.0 provides the primitive \language that is used to store the current language. When used with a pre-3.0 version this function has to be implemented by allocating a counter.

```
\def\bbl@version{⟨⟨version⟩⟩}
\def\bbl@date{⟨⟨date⟩⟩}
⟨⟨Define core switching macros⟩⟩
```

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

```
\adddialect #1#2{%
  \global\chardef#1#2\relax
  \bbl@usehooks{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\@gobble\bbl@tempa'
        set to \expandafter{\string\language\the\count@}. Reported}%
    \fi}
    \bbl@cs{languages}%
  \endgroup}
```

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

```
\def\bbl@ifname#1{%
  \begingroup
  \def\bbl@tempe{l@}%
  \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
  \bbl@tempd
  {\lowercase\expandafter{\bbl@tempd}%
    \{\uppercase\expandafter{\bbl@tempd}}%
    \@empty
    {\edef\bbl@tempd{\def\noexpand#1{#1}}%
      \uppercase\expandafter{\bbl@tempd}}%
  }
```

13
After a name has been 'fixed', the selectors will try to load the language. If even the fixed name is not defined, will load it on the fly, either based on its name, or if activated, its BCP47 code. We first need a couple of macros for a simple BCP 47 look up. It also makes sure, with \bbl@bcpcase, casing is the correct one, so that sr-Latn-Ba becomes fr-Latn-BA. Note #4 may contain some \@empty's, but they are eventually removed. \bbl@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}}{}% 
  \else \ifx\bbl@bcp\relax 
    \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}% 
  \fi \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}}{}% 
  \else \ifx\bbl@bcp\relax 
    \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}{\let\bbl@bcp\bbl@tempa}{}% 
  \fi \fi 
  \else \ifx\bbl@bcp\relax 
    \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}% 
  \fi \fi} 
\let\bbl@initoload\relax (-core) 
\def\bbl@provide@locale{% 
  \ifx\babelprovide\@undefined 
    \bbl@error{base-on-the-fly}{}{}{}% 
  \fi \fi 
\let\bbl@auxname\languagename % Still necessary. TODO 
\bbl@ifunset{bbl@bcp@map@\languagename}{}% Move uplevel?? 
\edef\languagename{\@nameuse{bbl@bcp@map@\languagename}}%
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.

\selectlanguage

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

\aftergroup

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

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

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 \xstring\aftergroup mechanism to help us. The command \aftergroup stores the token immediately following it to be executed when the current group is closed. So we define a temporary control sequence \bbl@pop@language to be executed at the end of the group. It calls \bbl@set@language with the name of the current language as its argument.
The previous solution works for one level of nesting groups, but as soon as more levels are used it is no longer adequate. For that case we need to keep track of the nested languages using a stack mechanism. This stack is called \bbl@language@stack and initially empty.

\begin{verbatim}
566 \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.

\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+}\
    \else
      \xdef\bbl@language@stack{\languagename+\bbl@language@stack}\
    \fi
  \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 \languagename. For this we first define a helper function.

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

\begin{verbatim}
579 \let\bbl@ifrestoring\@secondoftwo
580 \def\bbl@pop@language{\expandafter\bbl@pop@lang\bbl@language@stack\@@
581 \let\bbl@ifrestoring\@firstoftwo
582 \expandafter\bbl@set@language\expandafter{\languagename}}
\end{verbatim}

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

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

\begin{verbatim}
588 \chardef\localeid\z@
589 \def\bbl@id@last{0} % No real need for a new counter
590 \def\bbl@id@assign{\bbl@ifunset{bbl@id@@\languagename}{
591 \count@\bbl@id@last\relax
592 \advance\count@\@ne
593 \bbl@csarg\chardef{id@@\languagename}\count@}
594 \edef\bbl@id@last{\the\count@}
595 \ifcase\bbl@engine\or
596 \directlua{
597 \Babel = \Babel or {} \Babel.locale_props = \Babel.locale_props or {}
598 \Babel.locale_props{\bbl@id@last} = {}
599 \Babel.locale_props{\bbl@id@last}.name = '{\languagename'
\end{verbatim}
The unprotected part of \selectlanguage.

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

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

\def\BabelContentsFiles{toc,lof,lot} \def\bbl@set@language#1{% from selectlanguage, pop@ \% The old buggy way. Preserved for compatibility. \edef\languagename{% \ifnum\escapechar=\expandafter`\string#1\@empty \else\string#1\@empty\fi} \ifcat\relax\noexpand#1\% \expandafter\ifx\csname date\languagename\endcsname\relax \edef\languagename{#1}\% \let\localename\languagename \else \bbl@info{Using \string\language' instead of 'language' is\% deprecated. If what you want is to use a\% macro containing the actual locale, make\% sure it does not not match any language.\% Reported}\% \ifscantokens\undefined \def\localename{??}\% \else \scantokens\expandafter{\expandafter\def\expandafter\localename\expandafter{\languagename}}\% \fi \fi \else \bbl@info{Using \string\language' instead of 'language' is\% deprecated. If what you want is to use a\% macro containing the actual locale, make\% sure it does not not match any language.\% Reported}\% \ifscantokens\undefined \def\localename{??}\% \else \scantokens\expandafter{\expandafter\def\expandafter\localename\expandafter{\languagename}}\% \fi \fi \\select@language{\languagename}\% write to auxs \if@filesw \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant \bbl@savelastskip \protected@write\auxout{}{\string\babel@aux{\bbl@auxname}{}}\% \bbl@restorelastskip \fi \fi \bbl@usehooks{write}\% \fi \fi\%
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\langle lang\rangle command at definition time by expanding the \csname primitive.

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

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

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

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

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

At the end of the environment we need to switch off the extra definitions. The grouping mechanism of the environment will take care of resetting the correct hyphenation rules and “extras”. The `\foreignlanguage` command is another substitute for the `\selectlanguage` command. This command takes two arguments, the first argument is the name of the language to use for typesetting the text specified in the second argument. Unlike `\selectlanguage` this command doesn’t switch everything, it only switches the hyphenation rules and the extra definitions for the language specified. It does this within a group and assumes the `\extras{lang}` command doesn’t make any `\global` changes. The coding is very similar to part of `\selectlanguage`.

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

`\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).

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.
\begin{group}
\def\bbl@selectorname{foreign}\
\def\bbl@select@opts{#1}\
\let\BabelText@firstofone
\bbl@beforeforeign\
\foreign@language{#2}\
\bbl@usehooks{foreign}{}\
\BabelText{#3}% Now in horizontal mode!
\endgroup}

\begin{group}
\def\bbl@foreign@s#1#2{%
\begingroup
{\par}%
\def\bbl@selectorname{foreign*}\
\let\bbl@select@opts\@empty\let\BabelText\@firstofone
\foreign@language{#1}\
\bbl@usehooks{foreign*}{}\bbl@dirparastext
\BabelText{#2}% Still in vertical mode!
{\par}\
\endgroup}
\endgroup

\foreign@language
This macro does the work for \foreign@language 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.

\begin{group}
\def\foreign@language#1{%
% set name
\edef\languagename{#1}\
\ifbbl@usedategroup
\bbl@add\bbl@select@opts{,date,}\
\bbl@usedategroupfalse
\fi
\bbl@fixname\languagename% TODO. name@map here?
\bbl@provide@locale
\bbl@iflanguage\languagename{%
\let\bbl@select@type\@ne\bbl@switch{\languagename}}}
\endgroup

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

\begin{group}
\def\IfBabelSelectorTF#1{%
\bbl@xin@{,\bbl@selectorname,}{,\zap@space#1 \@empty,}\
\ifin@
\expandafter\@firstoftwo
\else
\expandafter\@secondoftwo
\fi\endgroup
\endgroup

\bbl@patterns
This macro selects the hyphenation patterns by changing the \language register. If special hyphenation patterns are available specifically for the current font encoding, use them instead of the default. It also sets hyphenation exceptions, but only once, because they are global (here language \lccode's has been set, too). \bbl@hyphenation@ is set to relax until the very first \babelhyphenation, so do nothing with this value. If the exceptions for a language (by its number, not its name, so that :ENC is taken into account) has been set, then use \hyphenation with both global and language exceptions and empty the latter to mark they must not be set again.

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

\begin{verbatim}
def\hyphenrules#1{% 
def\bbl@tempf{#1}% bl@fixname\bbl@tempf bl@iflanguage\bbl@tempf{% 
expandafter\bbl@patterns\expandafter{\bbl@tempf}% 
\ifx\languageshorthands\@undefined\else 
\languageshorthands{none}\relax
\else
\expandafter\expandafter\expandafter\set@hyphenmins
\csname\bbl@tempf hyphenmins\endcsname\relax
\fi}
\endgroup}
\end{verbatim}

\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.

\begin{verbatim}
def\providehyphenmins#1#2{%
\expandafter\csname\bbl@tempf hyphenmins\endcsname\relax
\@namedef{#1hyphenmins}{#2}%
}\end{verbatim}

\set@hyphenmins

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

\begin{verbatim}
def\set@hyphenmins#1#2{%
\lefthyphenmin#1\relax
\righthyphenmin#2\relax
}\end{verbatim}

\ProvidesLanguage

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

\begin{verbatim}
def\ProvidesLanguage#1[#2 #3 #4]{%
\end{verbatim}

22
\originalTeX The macro \originalTeX should be known to \TeX at this moment. As it has to be expandable we let it to $\@empty$ instead of $\relax$.

\if\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{\bbl@error{not-yet-available}{}{}{}}

\let\uselocale\setlocale
\let\locale\setlocale
\let\selectlocale\setlocale
\let\textlocale\setlocale
\let\textlanguage\setlocale
\let\languagetext\setlocale

4.2 Errors

\@nolanerr 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 (US) english, but it might also be empty.

\@noopterr When the package was loaded without options not everything will work as expected. An error message is issued in that case. When the format knows about \PackageError it must be \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@nullcaption{\protect\bbl@nullcaption@i}
\def\bbl@nullcaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
\global\namedef{#2}{\textbf{?#1?}}\protect@nameuse{#2}
\edef\bbl@tempa{#1}
\bbl@sreplace\bbl@tempa{name}{}%
\bbl@warning{\@backslashchar#1 not set for '\languagename'. Please,\%
define it after the language has been loaded\%
(typically in the preamble) with: \%
\string\setlocale\caption{\languagename}\{\bbl@tempa{..}\} \%
Feel free to contribute on github.com/latex3/babel.\%
Reported}}
\def\bbl@tentative{\protect\bbl@tentative@i}
\def\bbl@tentative@i#1{%
\bbl@warning{Some functions for '\#1' are tentative.\%}
They might not work as expected and their behavior could change in the future. Reported}}

\def\nolanerr#1{\bbl@error{undefined-language}{#1}{}{}}
\def\nopatterns#1{\bbl@warning{No hyphenation patterns were preloaded for the language '#1' into the format. Please, configure your TeX system to add them and rebuild the format. Now I will use the patterns preloaded for \nulllanguage\space instead}}
\let\bbl@usehooks\@gobbletwo
\ifx\bbl@onlyswitch\@empty\endinput\fi
% Here ended switch.def
Here ended the now discarded switch.def. Here also (currently) ends the base option.

\ifx\directlua\@undefined\else
\ifx\bbl@luapatterns\@undefined
\input luababel.def
\fi\fi
\bbl@trace{Compatibility with language.def}
\ifx\bbl@languages\@undefined
\ifx\directlua\@undefined
\openin1 = language.def % TODO. Remove hardcoded number
\ifeof1
\closein1
\message{I couldn't find the file language.def}
\else
\closein1
\begingroup
\def\addlanguage#1#2#3#4#5{\expandafter\ifx\csname lang@#1\endcsname\relax\else\global\expandafter\let\csname l@#1\expandafter\endcsname\csname lang@#1\endcsname\fi}\def\uselanguage#1{}\input language.def\endgroup
\fi
\fi\fi
\chardef\l@english\z@
\fi
\addto\Ittakestwoarguments,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.
\def\addto#1#2{%\ifx#1\@undefined\def#1{#2}%\else\ifx#1\relax\def#1{#2}%\else{\toks@\expandafter{#1#2}%\xdef#1{\the\toks@}}%\fi\fi\fi\endgroup
\def\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.
To redefine a command, we save the old meaning of the macro. Then we redefine it to call the 
original macro with the 'sanitized' argument. The reason why we do this way is that we don't want 
to redefine the \TeX macros completely in case their definitions change (they have changed in the past). A macro named \macro will be saved new control sequences named \orgmacro.

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

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

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

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

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

4.3 Hooks

Admittedly, the current implementation is somewhat simplistic and does very little to catch errors, 
but it is meant for developers, after all. \bbl@usehooks is the commands used by babel to execute 
hooks defined for an event.
To ensure forward compatibility, arguments in hooks are set implicitly. So, if a further argument is added in the future, there is no need to change the existing code. Note events intended for hyphen.cfg are also loaded (just in case you need them for some reason).

```latex
\def\bbl@evargs{% <- don't delete this comma
  everylanguage=1, loadkernel=1, loadpatterns=1, loadexceptions=1, %
  adddialect=2, patterns=2, defaultcommands=0, encodedcommands=2, write=0, %
  beforeextras=0, afterextras=0, stopcommands=0, stringprocess=0, %
  hyphenation=2, initiateactive=3, afterreset=0, foreign=0, foreign*=0, %
  beforestart=0, languagename=2, begindocument=1} %
\ifx\NewHook\@undefined\else % Test for Plain (?)\fi
\def\bbl@tempa#1=#2\@@\{}{\NewHook{babel/#1}} %
\bbl@foreach\bbl@evargs{\bbl@tempa#1\@@}\{}
\fi
```

\texttt{\bbl@evargs} The user command just parses the optional argument and creates a new macro named \texttt{\bbl@e@\langle language\rangle}. We register a hook at the afterextras event which just executes this macro in a "complete" selection (which, if undefined, is \texttt{\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 \texttt{\bbl@e@\langle language\rangle} contains \texttt{\bbl@ensure{\langle include\rangle}{\langle exclude\rangle}{\langle fontenc\rangle}}, which in in turn loops over the macros names in \texttt{\bbl@captionslist}, excluding (with the help of \texttt{\in@}) those in the exclude list. If the fontenc is given (and not \texttt{\relax}), the \texttt{\fontencoding} is also added. Then we loop over the include list, but if the macro already contains \texttt{\foreignlanguage}, nothing is done.

Note this macro (1) is not restricted to the preamble, and (2) changes are local.

```
\def\bbl@trace{Defining babelensure}
\newcommand\babelensure[2][]{% 1: include 2: exclude 3: fontenc
\def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
\ifx##1\@undefined % 3.32 - Don't assume the macro exists
\edef##1{\noexpand\bbl@nocaption{\bbl@stripslash##1}{\languagename\bbl@stripslash##1}}%
\fi
\ifx##1\@empty\else
\in@{##1}{#2}\
\ifin@\else
\bbl@ifunset{bbl@ensure@\languagename}\{
\\DeclareRobustCommand\bbl@ensure@\languagename[1]{% 1: include 2: exclude 3: fontenc
\foreignlanguage{\languagename}{\ifx\relax#3\else\fontencoding{#3}\selectfont\fi}}
\bbl@exp{\bbl@ensure@\languagename}{#1}{#2}{#3}
\bbl@exp{\bbl@ensure@\languagename}{#1}{#2}{#3}
\fi
```

```
```
4.4 Setting up language files

The \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 \relax we are dealing with a control sequence which we can compare with \@undefined. If so, we call \ldf@quit to set the main language, restore the category code of the @-sign and call \endinput

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

\bbl@trace{Macros for setting language files up}
\def\bbl@ldfinit{%
\let\bbl@screset\@empty
\let\BabelStrings\bbl@opt@string
\let\BabelOptions\@empty
\let\BabelLanguages\relax
\ifx\originalTeX\@undefined
\else
\fi
\def\originalTeX{
\fi}
\def\ldf@init#1#2{%
\chardef\atcatcode=\catcode`@
\let\catcode`@=11\relax
\chardef\eqcatcode=\catcode`=
\let\catcode`\@=12\relax
\expandafter\if\expandafter\backslashchar\string#2\@nil
\expandafter\car\string#2\@nil

27
\ldfquit \ If \ this \ macro \ interrupts \ the \ processing \ of \ a \ language \ definition \ file.  
\ldfquit \ \def\ldfquit#1{\expandafter\main@language\expandafter{#1}\catcode`@=\atcatcode \let\atcatcode\relax \catcode`==\eqcatcode \let\eqcatcode\relax \endinput}  
\ldfquit \ This \ macro \ interrupts \ the \ processing \ of \ a \ language \ definition \ file.  
\ldfquit \ 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.  
\ldfquit \ \def\bbl@afterldf#1{% TODO. \ Merge \ into \ the \ next \ macro? \ Unused \ elsewhere  
\bbl@afterlang  
\let\bbl@afterlang\relax  
\let\BabelModifiers\relax  
\let\bbl@screset\relax}  
\ldfquit \ \def\ldf@finish#1{\loadlocalcfg{#1}\bbl@afterldf{#1}\expandafter\main@language\expandafter{#1}\catcode`@=\atcatcode \let\atcatcode\relax}  
\ldfquit \ After \ the \ preamble \ of \ the \ document \ the \ commands \ \ldfinit, \ \ldfquit \ and \ \ldffinish \ are \ no \ longer \ needed. \ Therefore \ they \ are \ turned \ into \ warning \ messages \ in \ \LaTeX.  
\ldfquit \ \@onlypreamble\ldfinit  
\ldfquit \ \@onlypreamble\ldfquit  
\ldfquit \ \@onlypreamble\ldffinish  
\ldfquit \ 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.  
\ldfquit \ \def\main@language#1{\def\bbl@main@language{#1}\let\languagename\bbl@main@language % TODO. \ Set \ localename  
\bbl@id@assign  
\bbl@patterns{\languagename}}  
\ldfinit \ We \ also \ have \ to \ make \ sure \ that \ some \ code \ gets \ executed \ at \ the \ beginning \ of \ the \ document, \ either \ when \ the \ aux \ file \ is \ read \ or, \ if \ it \ does \ not \ exist, \ when \ the \ \AtBeginDocument \ is \ executed. \ Languages \ do \ not \ set \ \pagedir, \ so \ we \ set \ here \ for \ the \ whole \ document \ to \ the \ main \ \bodydir.  
\ldfinit \ \def\bbl@beforesort{%  
\def\@nolanerr##1{\bbl@warning{Undefined \ language \ '#1' \ in \ aux. \ \Reported}}\%  
\bbl@usehooks{beforesort}\%  \global\let\bbl@beforesort\relax\%  \AtBeginDocument{%  
\{
ameuse{bbl@beforesort}\% \ Group!  
\if@filesiw  
\providecommand\babel@aux[2]{%  
\immediate\write\maiaux{%  
\string\providecommand\string\babel@aux[2]{}}%  
\endinput}
A bit of optimization. Select in heads/foots the language only if necessary.

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

4.5 Shorthands

The macro \bbl@add@special is used to add a new character (or single character control sequence) to the macro \dospecials (and \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.

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.
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 \active@char{char} by default (\char being the character to be made active). Later its definition can be changed to expand to \active@char{char} by calling \bbl@activate{char}.

For example, to make the double quote character active one could have \initiate@active@char{"} in a language definition file. This defines " as \active@prefix \active@char{"} (where the first " is the character with its original catcode, when the shorthand is created, and \active@char{"} is a single token). In protected contexts, it expands to \protect " or \noexpand " (ie, with the original "); otherwise \active@char{"} is executed. This macro in turn expands to \normal@char" in "safe" contexts (eg, \label), but \user@active" in normal "unsafe" ones. The latter search a definition in the user, language and system levels, in this order, but if none is found, \normal@char" is used. However, a deactivated shorthand (with \bbl@deactivate is defined as \active@prefix \normal@char".

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

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.

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

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).

Now we have set \normal@char{⟨char⟩}, we must define \active@char{⟨char⟩}, to be executed when the character is activated. We define the first level expansion of \active@char{⟨char⟩} to check the status of the @safe@actives flag. If it is set to true we expand to the ‘normal’ version of this character, otherwise we call \user@active{⟨char⟩} to start the search of a definition in the user, language and system levels (or eventually \normal@char{⟨char⟩}).
We now define the default values which the shorthand is set to when activated or deactivated. It is set to the deactivated form (globally), so that the character expands to

\active@prefix (char) \normal@char (char)

(where \active@char is one control sequence!).

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.

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

Finally, a couple of special cases are taken care of. (1) If we are making the right quote (‘) active we need to change \prime 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.

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

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.
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.

\begin{verbatim}
def\bbl@sh@select#1#2{%\expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax\bbl@afterelse\bbl@scndcs\else\bbl@afterfi\csname#1@sh@#2@sel\endcsname\fi}
\end{verbatim}

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

\begin{verbatim}
\begingroup\bbl@ifunset{ifincsname}% TODO. Ugly. Correct? Only Plain?\{\gdef\active@prefix#1{%\ifx\protect\@typeset@protect\else\ifx\protect\@unexpandable@protect\noexpand#1\%\else\protect#1\%\fi\expandafter\@gobble\fi\fi\expandafter\expandafter\expandafter\@gobble\fi\fi\}
\end{verbatim}

\if@safe@actives In some circumstances it is necessary to be able to reset the shorthand to its ‘normal’ value (usually the character with catcode ’other’) on the fly. For this purpose the switch @safe@actives is available. The setting of this switch should be checked in the first level expansion of \active@char(char). When this expansion mode is active (with \@safe@activestrue), something like ”13” becomes ”12” in an \edef (in other words, shorthands are \string’ed). This contrasts with \protected@edef, where catcodes are always left unchanged. Once converted, they can be used safely even after this expansion mode is deactivated (with \@safe@activesfalse).

\begin{verbatim}
\newif\if@safe@actives\@safe@activesfalse\bbl@restore@actives
\end{verbatim}

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

\begin{verbatim}
\def\bbl@restore@actives{%\if@safe@actives\@safe@activesfalse\fi}
\end{verbatim}
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(ch)` in the case of `\bbl@activate`, or `\normal@char(ch)` in the case of `\bbl@deactivate`.

```
\chardef\bbl@activated\z@
\def\bbl@activate#1{%\chardef\bbl@activated\@ne %\bbl@withactive{\expandafter\let\expandafter}#1% \csname bbl@active@\string#1\endcsname}
\def\bbl@deactivate#1{%\chardef\bbl@activated\tw@ %\bbl@withactive{\expandafter\let\expandafter}#1% \csname bbl@normal@\string#1\endcsname}
```

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

```
\def\bbl@firstcs#1#2{\csname#1\endcsname}
\def\bbl@scndcs#1#2#3#4{\csname#1#2\endcsname}
```

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

```
\def\babel@texpdf#1#2#3#4{%\ifx\texorpdfstring\@undefined \textormath{#1}{#3}% \else \texorpdfstring{\textormath{#1}{#3}}{#2}% \fi}
```

```
% \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
% \def\@decl@short#1#2#3#4{\def\bbl@tempa{#3} %\ifx\bbl@tempa\@empty \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs %\bbl@ifunset{#1@sh@\string#2@}{}% {\def\bbl@tempa{#4} %\expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa %\else \bbl@info %\{Redefining #1 shorthand \string#2\% in language \CurrentOption\% %\fi}% \@namedef{#1@sh@\string#2@}{#4}% \else \expandafter\let\csname #1@sh@\string#2@\string#3@\endcsname\bbl@tempa %\bbl@ifunset{#1@sh@\string#2@\string#3@}{}% {\def\bbl@tempa{#4} %\expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa %\else \bbl@info %\{Redefining #1 shorthand \string#2\string#3\% in language \CurrentOption\% %\fi}% \@namedef{#1@sh@\string#2@\string#3@}{#4}% \fi%}
```

34
Some of the shorthands that will be declared by the language definition files have to be usable in both text and mathmode. To achieve this the helper macro \textormath is provided.

\begin{verbatim}
def\textormath{\ifmmode\expandafter\@secondoftwo\else\expandafter\@firstoftwo\fi}
def\user@group{user}
def\language@group{english} % TODO. I don't like defaults
def\system@group{system}
def\useshorthands{\@ifstar\bbl@usesh@s{\bbl@usesh@x{}}}
def\bbl@usesh@x#1{\bbl@usesh@x{\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}#1}
def\bbl@usesh@x#1#2{\bbl@ifshorthand{#2}{\def\user@group{user}\initiate@active@char{#2}#1}{\bbl@error{shorthand-is-off}{}{#2}{}}}
def\defineshorthand{ currently we only support two groups of user level shorthands, named internally user and user@<lang> (language-dependent user shorthands). By default, only the first one is taken into account, but if the former is also used (in the optional argument of \defineshorthand) a new level is inserted for it (user@generic, done by \bbl@set@user@generic); we make also sure {} and \protect are taken into account in this new top level.
def\languageshorthands#1{\def\language@group{#1}}
\end{verbatim}

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

\user@group 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.

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

\system@group 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].

\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].
\aliasshorthand \textit{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*.

\begin{verbatim}
1435 \def\aliasshorthand#1#2{%
1436  \bbl@ifshorthand{#2}%
1437   {\expandafter{\csname active@char\string#2\endcsname}\relax
1438    \ifx\document@notprerr
1439     \@notshorthand{#2}%
1440   \else
1441     \initiate@active@char{#2}%
1442     \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1443     \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1444     \bbl@activate{#2}%
1445   \fi
1446  \fi}
1447 {\bbl@error{shorthand-is-off}{\{}(#2){}\}}
\end{verbatim}

\@notshorthand

\begin{verbatim}
1448 \def\@notshorthand#1{\bbl@error{not-a-shorthand}{#1}{}{}}
\end{verbatim}

\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.

\begin{verbatim}
1449 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1450 \DeclareRobustCommand*\shorthandoff{%
1451   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1452 \def\bbl@shorthandoff#1#2{\bbl@switch@sh#1#2\@nnil}
\end{verbatim}

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

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

\begin{verbatim}
1453 \def\bbl@switch@sh#1#2{%
1454  \ifx#2\@nnil\else
1455   \bbl@ifunset{bbl@active@char\string#2}{%\bbl@error{not-a-shorthand-b}{\{}#2{}}\}%
1456   {\ifcase#1% off, on, off*
1457     \catcode`#212\relax
1458     \or
1459     \catcode`#2\active
1460     \bbl@ifunset{bbl@shdef@char\string#2}{%\bbl@error{not-a-shorthand-b}{\{}#2{}}\}%
1461     \unless\bbl@active@char\string#2
1462       \csname bbl@oridef@char\string#2\endcsname
1463       \csname bbl@oricat@char\string#2\endcsname
1464     \else
1465       \bbl@activate{#2}\
1466     \fi
1467   \fi
1468 \fi
1469 \fi}{\bbl@error{not-a-shorthand-b}{\{}#2{}}\}%
1470 \fi
1471 \fi}
\end{verbatim}
Note the value is that at the expansion time; eg, in the preamble shorthands are usually deactivated.

\def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
\def\bbl@putsh#1{% 
  \ifx\empty@\empty@nil{}
  \csname bbl@active@string#1\endcsname}
\ifx\empty@\empty@nil{}
  \csname language@group @sh\string#1\endcsname}
\ifx\empty@\empty@nil{}
  \string#2@\fi\endcsname}
\ifx\empty@\empty@nil{}
  \csname
code@group @sh@string#1@% 
  \ifx\empty@#2\else\string#2@\fi\endcsname}
\ifx\empty@\empty@nil{}
  \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{% 
  \ifx#2\empty@nil{}
  \bbl@afterfi
  \bbl@afterelse\expandafter\@firstoftwo
  \else
  \expandafter\@secondoftwo
\fi
\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 or off.

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

One of the internal macros that are involved in substituting \prime for each right quote in mathmode is \prim@. 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.

\def\bbl@prim@s{\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
\let\bbl@pr@m@s={
\bbl@if@primes*\pr@@@t\egroup}}
\endgroup

Usually the ~ is active and expands to \penalty\@M\&. When it is written to the .aux file it is written expanded. To prevent that and to be able to use the character ~ as a start character for a shorthand, it is redefined here as a one character shorthand on system level. The system declaration is in most cases redundant (when ~ is still a non-break space), and in some cases is inconvenient (if ~ has been redefined); however, for backward compatibility it is maintained (some existing documents may rely on the babel value).
\begin{verbatim}
\def\OT1dqpos{127}
\def\T1dqpos{4}
\ifx\f@encoding\@undefined
\def\f@encoding{OT1}
\fi

4.6 Language attributes

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

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.

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.

\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 `\AtBeginDocument` because the attributes are set in the document preamble, after \bbl 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.

\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.

\bbl@clear@tribs This macro removes all the attribute code from \TeX’s memory at \begin{document} time (if any is present).

4.7 Support for saving macro definitions

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

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

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

\begin{verbatim}
\def\babel@save#1{%  
  \def\bbl@tempa{{,#1,}}% Clumsy, for Plain  
  \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{\expandafter{\expandafter,\bbl@savedextras,}}%  
  \expandafter\in@\bbl@tempa  
  \ifin@  
    \bbl@add\bbl@savedextras{,#1,}%  
    \bbl@carg\let{babel@\number\babel@savecnt}#1\relax  
    \toks@\expandafter{\originalTeX\let#1=}%  
    \bbl@exp{\def\\originalTeX{\the\toks@\langle\texttt{babel@\number\babel@savecnt}>\relax}}%  
  \advance\babel@savecnt\@ne  
  \fi}  
\def\babel@savevariable#1{%
  \toks@\expandafter{\originalTeX \let#1=}%  
  \bbl@exp{\def\\originalTeX{\the\toks@\the#1\relax}}
\end{verbatim}

Some languages need to have \texttt{\frenchspacing} in effect. Others don't want that. The command \texttt{\bbl@frenchspacing} switches it on when it isn't already in effect and \texttt{\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 \texttt{\ babelprovide}. This new method should be ideally the default one.

\begin{verbatim}
\def\bbl@frenchspacing{%  
  \ifnum\the\sfcode`\string.\@m  
    \let\bbl@nonfrenchspacing\relax  
  \else  
    \frenchspacing  
    \let\bbl@nonfrenchspacing
onfrenchspacing  
  \fi}  
\let\bbl@nonfrenchspacing\nonfrenchspacing  
\let\bbl@elt\relax  
\edef\bbl@fs@chars{\bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}  
\bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}  
\bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
\def\bbl@pre@fs{\bbl@save@sfcodes}  
\def\bbl@post@fs{\bbl@save@sfcodes\bbl@cl{frspc}\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}  
\if u\bbl@tempa % do nothing  
  \else\if n\bbl@tempa % non french  
    \bbl@elt##1##2##3{\ifnum\sfcode`##1=##2\relax  
    \babel@savevariable{\sfcode`##1}  
\end{verbatim}

\texttt{\original\TeX} has to be expandable, i.e. you shouldn't let it to \texttt{\relax}.
4.8 Short tags

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

4.9 Hyphens

\texttt{\textbackslash babelhyphenation} This macro saves hyphenation exceptions. Two macros are used to store them: \texttt{\textbackslash bbl@hyphenation@} for the global ones and \texttt{\textbackslash bbl@hyphenation<lang>} for language ones. See \texttt{\textbackslash bbl@patterns} above for further details. We make sure there is a space between words when multiple commands are used.
This macro makes hyphenation possible. Basically its definition is nothing more than \nobreak \hskip 0pt plus 0pt\%.

\def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
\def\bbl@t@one{T1}
\def\allowhyphens{\ifx\cf@encoding\bbl@t@one\else\bbl@allowhyphens\fi}
\def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
\def\bbl@hyphen{\@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
\def\bbl@hyphen@i#1#2{\bbl@ifunset{bbl@hy@#1#2\@empty}{\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}\bbl@hy@#1#2\@empty\@empty}
\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}
\def\bbl@hyphenchar{\ifnum\hyphenchar\font=\m@ne\babelnullhyphen\else\char\hyphenchar\font\fi}
\def\bbl@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bbl@allowhyphens}

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

\textbf{bbl@allowhyphens} This macro makes hyphenation possible. Basically its definition is nothing more than \nobreak \hskip 0pt plus 0pt\%

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

\newcommand\babelnullhyphen{\char\hyphenchar\font}
\def\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}
\def\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}
\def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
4.10 Multiencoding strings

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

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

1731 \bbl@trace{Multiencoding strings}
1732 \def\bbl@toglobal#1{\global\let#1#1}

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

1733 \langle\langle \text{More package options} \rangle\rangle ≡
1734 \DeclareOption{nocase}{}
1735 \langle\langle / \text{More package options} \rangle\rangle

The following package options control the behavior of \SetString.

1736 \langle\langle \text{More package options} \rangle\rangle ≡
1737 \let\bbl@opt@strings\@nnil % accept strings=value
1738 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1739 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1740 \def\BabelStringsDefault{generic}
1741 \langle\langle / \text{More package options} \rangle\rangle

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.

1742 \@onlypreamble\StartBabelCommands
1743 \def\StartBabelCommands{%
1744 \begingroup
1745 \@tempcnta=\string\^^M 7F
1746 \def\bbl@tempa{%
1747 \ifnum\@tempcnta=\string\^^M FF\else
1748 \catcode\@tempcnta=11
1749 \advance\@tempcnta\@ne
1750 \expandafter\bbl@tempa
1751 \fi%
1752 \bbl@tempa
1753 \langle\langle \text{Macros local to BabelCommands} \rangle\rangle
1754 \def\bbl@procstring##1##2{%
1755 \providecommand##1{##2}%
1756 \bbl@toglobal##1%
1757 \global\let\bbl@scafter\@empty
1758 \let\StartBabelCommands\bbl@startcmds
1759 \ifx\BabelLanguages\relax
1760 \let\BabelLanguages\CurrentOption
1761 \fi
1762 \begingroup
1763 \langle\langle \text{Macros local to BabelCommands} \rangle\rangle
1764 \def\bbl@startcmds{%
1765 \ifx\bbl@opt@strings\@nnil % local flag - disable 1st stopcommands
1766 \let\bbl@opt@strings\relax\else
1767 \\bbl@usehooks{stopcommands}{}%
1768 \fi
1769 \endgroup
1770 \begingroup
1771 \@ifstar
1772 \langle\langle \text{Macros local to BabelCommands} \rangle\rangle
1773 \def\bbl@startcmds{%
1774 \fi
1775 \bbl@startcmds@i%
1776 \bbl@startcmds@i%
Parse the encoding info to get the label, input, and font parts.
Select the behavior of \SetString. There are two main cases, depending on 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 string 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.
There are two versions of `bbl@scswitch`. The first version is used when `ldfs` are read, and it makes sure `⟨group⟩⟨language⟩` is reset, but only once (`bbl@screset` is used to keep track of this). The second version is used in the preamble and packages loaded after `babel` and does nothing. The macro `\bbl@forlang` loops `\bbl@L` but its body is executed only if the value is in `\BabelLanguages` (inside `babel`) or `\date⟨language⟩` is defined (after `babel` has been loaded). There are also two versions of `\bbl@forlang`. The first one skips the current iteration if the language is not in `\BabelLanguages` (used in `ldfs`), and the second one skips undefined languages after `babel` has been loaded.

```
def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@in@{,#1,}{,\BabelLanguages,}\
\ifin@#2\relax\fi}}
def\bbl@scswitch{\bbl@forlang\bbl@tempa{\bbl@G\@empty\else\bbl@if\SetString\@gobbletwo\else\edef\bbl@GL{\bbl@G\bbl@tempa}\bbl@in@{,\bbl@GL,}{,\bbl@screset,}\bbl@if\else\global\expandafter\let\csname\bbl@GL\endcsname\@undefined\xdef\bbl@screset{\bbl@screset,\bbl@GL}\fi\fi}}
```

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

**Strings**  
The following macro is the actual definition of `\SetString` when it is “active”. First save the “switcher”. Create it if undefined. Strings are defined only if undefined (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{%\eg, \prefacename{<string>}\bbl@forlang\bbl@tempa{%\edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}\bbl@if\@empty\else\edef\bbl@GL{\bbl@LC}\bbl@if\else\global\expandafter\let\csname\bbl@GL\endcsname\@undefined\xdef\bbl@screset{\bbl@screset,\bbl@GL}\fi\fi}}
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's used in `\setlocalecaption`.

45
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.

\def\bbl@scset#1#2{\def#1{#2}}
\def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}
\count@\z@
\bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
\advance\count@\@ne
\toks@\expandafter{\bbl@tempa}%
\bbl@exp{\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
\count@=\the\count@\relax}}%

\newcommand\SetString[2]{% one to one.
\ifnum\lccode#1=#2\else
\ babel@savevariable{\lccode#1}%
\lccode#1=#2\relax
\fi}
\newcommand\BabelLowerMM[4]{% many-to-many
\@tempcnta=#1\relax
\@tempcntb=#4\relax
\def\bbl@tempa{%
\ifnum\@tempcnta>#2\else
\@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
\advance\@tempcnta#3\relax
\fi}}
The following package options control the behavior of hyphenation mapping.

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

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.
4.11 Macros common to a number of languages

\set@low@box

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

\save@sf@q

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

4.12 Making glyphs available

This section makes a number of glyphs available that either do not exist in the OT1 encoding and have to be 'faked', or that are not accessible through T1enc.dfs.

4.12.1 Quotation marks

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

\quotesinglbase We also need the single quote character at the baseline.
Make sure that when an encoding other than OT1 or T1 is used this glyph can still be typeset.

\ProvideTextCommandDefault{\quotesinglebase}{{OT1}}
\UseTextSymbol{OT1}{\quotesinglebase}

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

\guillemetright
\guilsinglleft
\guilsinglright

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

\guilsinglleft
\guilsinglright

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

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

\IJ  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).

\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.

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.
\grq

\ProvideTextCommandDefault{\glq}{\textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}  

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

\ProvideTextCommand{\grq}{T1}{\textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}  
\ProvideTextCommand{\grq}{TU}{\textormath{\textquoteleft}{\mbox{\textquoteleft}}}  
\ProvideTextCommand{\grq}{OT1}{\save@sf@q{\kern-.0125em\textormath{\textquoteleft}{\mbox{\textquoteleft}}\kern.07em\relax}}  
\ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}

\glqq The 'german' double quotes.
\grqq

\ProvideTextCommandDefault{\glqq}{\textormath{\quotedblbase}{\mbox{\quotedblbase}}}  

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

\ProvideTextCommand{\grqq}{T1}{\textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}  
\ProvideTextCommand{\grqq}{TU}{\textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}  
\ProvideTextCommand{\grqq}{OT1}{\save@sf@q{\kern-.07em\textormath{\textquotedblleft}{\mbox{\textquotedblleft}}\kern.07em\relax}}  
\ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}

\flq The 'french' single guillemets.
\frq

\ProvideTextCommandDefault{\flq}{\textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}  
\ProvideTextCommandDefault{\frq}{\textormath{\guilsinglright}{\mbox{\guilsinglright}}}  
\ProvideTextCommandDefault{\flqq}{\textormath{\guillemetleft}{\mbox{\guillemetleft}}}  
\ProvideTextCommandDefault{\frqq}{\textormath{\guillemetright}{\mbox{\guillemetright}}}

4.12.4 Umlauts and tremas

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

\umlauthigh\umlautlow

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

\def\umlauthigh{%  
\def\bbl@umlauta##1{\leavevmode\bgroup\accent\csname f@encoding dqpos \endcsname##1\bbl@allowhyphens\egroup}  
\let\bbl@umlaute=\bbl@umlauta}  
\def\umlautlow{%  
\def\bbl@umlauta{\protect\lower@umlaut}  
\def\bbl@umlaute{\protect\lower@umlaut}  
\umlauthigh

51
\lower@umlaut \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 \langle dimen \rangle register.

\def\lower@umlaut#1{\leavevmode\bgroup\leavevmode\U@D 1ex% \setbox\z@\hbox{\char\csname f@encoding dqpos\endcsname}% \dimen@ -.45ex\advance\dimen@\ht\z@ \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}% \accent\csname f@encoding dqpos\endcsname \fontdimen5\font\U@D #1% \egroup}

\AtBeginDocument{\def\lower@umlaut#1{\leavevmode\bgroup\leavevmode\U@D 1ex% \setbox\z@\hbox{\char\csname f@encoding dqpos\endcsname}% \dimen@ -.45ex\advance\dimen@\ht\z@ \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}% \accent\csname f@encoding dqpos\endcsname \fontdimen5\font\U@D #1% \egroup}

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

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 \texttt{BibLaTeX} only. \texttt{bbl} provide 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.

53
\let\bbl@extend@ini@gobble

\newcommand\babelprovide[2][{}]{%
\let\bbl@savelangname=languagename
\edef\bbl@savelocaleid{\the\localeid}%
% Set name and locale id
\edef\languagename{#2}%
\bbl@id@assign
% Initialize keys
\bbl@vforeach{captions,date,import,main,script,language,%
    hyphenrules,linebreaking,justification,mapfont,maparabic,%
    mapdigits,intraspace,intrapenalty,onchar,transforms,alph,%
    Alph,labels,labels*,calendar,date,casing,interchar}%
\global\let\bbl@release@transforms=@empty
\global\let\bbl@release@casing=@empty
\let\bbl@calendars=@empty
\global\let\bbl@inidata=@empty
\global\let\bbl@extend@ini@gobble
\global\let\bbl@included@inis=@empty
\gdef\bbl@key@list{;}
\bbl@forkv{#1}{%
    \in@{/}{##1}% With /, (re)sets a value in the ini
    \ifin@
    \global\let\bbl@extend@ini@gobble\bbl@extend@ini@aux
    \bbl@renewinikey##1\@@{##2}%
    \else
    \bbl@csarg\ifx{KVP@##1}\@nnil\else
    \bbl@error{unknown-provide-key}{##1}{}{}%
    \fi
    \bbl@csarg\def{KVP@##1}{##2}%
    \fi}
\chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
\bbl@ifunset{date#2}\z@\{\bbl@ifunset{bbl@llevel@#2}\tw@\%
% == init ==
\if\bbl@screset\@undefined
\bbl@ldfinit
\fi
% == date (as option) ==
% \ifx\bbl@KVP@date\@nnil\else
% \fi
% ==
\let\bbl@lbkflag\relax % \@empty = do setup linebreak, only in 3 cases:
\ifcase\bbl@howloaded
\let\bbl@lbkflag@relay\relax % \@empty = do setup linebreak, only in 3 cases:
\ifcase\bbl@howloaded
\let\bbl@lbkflag@empty \empty % new
\else
\if\bbl@KVP@hyphenrules\@nnil\else
\let\bbl@lbkflag@empty \empty
\fi
\if\bbl@KVP@import\@nnil\else
\let\bbl@lbkflag@empty \empty
\fi
\fi
% == import, captions ==
% \if\bbl@KVP@import\@nnil\else
% \fi
% ==
\let\bbl@exp{\\bbl@ifblank{\bbl@KVP@import}{%
\begingroup
\def\BabelBeforeIni##1##2{%\gdef\bbl@KVP@import{##1}\endinput
\bbl@input@texini{#2}\endgroup}
%
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.
\bbl@ifunset{bbl@chrng@\languagename}{}%
\directlua{
Babel.set_chranges_b('\bbl@cl{sbcp}', '\bbl@cl{chrng}') }
\fi
% == onchar ==
\ifx\bbl@KVP@onchar\@nnil\else
\bbl@luahyphenate
\bbl@exp{\
AddToHook{env/document/before}{\select@language{#2}{}}}%
\directlua{
if Babel.locale_mapped == nil then
  Babel.locale_mapped = true
  Babel.linebreaking.add_before(Babel.locale_map, 1)
  Babel.loc_to_scr = {}
  Babel.chr_to_loc = Babel.chr_to_loc or {}
end
Babel.locale_props[\the\localeid].letters = false
}%
\bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
\ifi% Needed if bbl@onchar has a command associated with it
\bbl@exp{\
AddBabelHook{babel-onchar}{beforestart}{{\bbl@starthyphens}}}%
\fi
\bbl@add\bbl@starthyphens{\bbl@patterns@lua{\languagename}}%
% TODO - error/warning if no script
\directlua{
if Babel.script_blocks['\bbl@cl{sbcp}'] then
  Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbl@cl{sbcp}']
  Babel.locale_props[\the\localeid].lg = \the@nameuse{l@\languagename}\space
end
}%
\bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
\ifi
\bbl@ifndef{bbl@starthyphens}{\bbl@starthyphens} % Needed if no explicit selection
  \AddBabelHook{bbl-starthyphens}{beforestart}{\bbl@starthyphens}%
\fi
\bbl@exp{\bbl@add\bbl@starthyphens
{\bbl@patterns@lua{\languagename}}%}
% TODO - error/warning if no script
\directlua{
if Babel.script_blocks['\bbl@cl{sbcp}'] then
  Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbl@cl{sbcp}']
  Babel.locale_props[\the\localeid].lg = \the@nameuse{l@\languagename}\space
end
}%
\bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
\ifi
\bbl@ifndef{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}%}
\bbl@ifndef{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}%}
\directlua{
  if Babel.script_blocks['\bbl@cl{sbcp}'] then
    Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbl@cl{sbcp}']
  end}
\ifi\bbl@mapselect\@undefined % TODO. almost the same as mapfont
  \AtBeginDocument{%
    \bbl@patchfont{\bbl@mapselect}{\selectfont}%
  }%
\def\bbl@mapselect{%
  \let\bbl@mapselect\relax
  \edef\bbl@prefontid{\fontid\font}
}%
\def\bbl@mapdir##1{%
  \begingroup
    \setbox\z@\hbox{% Force text mode
      \def\languagename{##1}%
      \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
      \bbl@switchfont
      \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
\directlua{
Babel.locale_props[the\csname bbl@id@#1\endcsname]%
    ["/bbl@prefontid"] = \fontid\font\space}%
\fi}%
\endgroup}%
\fi}
\bbl@exp{\\bbl@add\\bbl@mapselect\\\bbl@mapdir\languagename}}%
\fi
% TODO - catch non-valid values
\fi
% == mapfont ==
% For bidi texts, to switch the font based on direction
%\if\bbl@KVP@mapfont\@nnil\else
\bbl@ifsamestring\bbl@KVP@mapfont{direction}{}% 
{\bbl@error{unknown-mapfont}{}{}{}}%
\bbl@ifunset{bbl@lsys\languagename}{\bbl@provide@lsys\languagename}{}%
\bbl@ifunset{bbl@wdir\languagename}{\bbl@provide@dirs\languagename}{}%
%\if\bbl@mapselect\@undefined
% TODO. See onchar.
\AtBeginDocument{
\bbl@patchfont{\bbl@mapselect{\bbl@mapdir\languagename}}% 
\def\bbl@mapselect{\selectfont}
\def\bbl@mapdir##1{\def\languagename{##1}\
\let\bbl@ifrestoring\@firstoftwo % avoid font warning
\bbl@switchfont
\directlua{Babel.fontmap\[\bbl@wdir\languagename\]%=\fontid\font}}%
\fi
\bbl@exp{\\bbl@add\\bbl@mapselect\\\bbl@mapdir\languagename}%
% == Line breaking: intraspace, intrapenalty ==
% For CJK, East Asian, Southeast Asian, if interspace in ini
%\if\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
%\if\bbl@engine\or
%\bbl@xin{/c}{/bbl@cl{lnbrk}}%
%\fi
%\bbl@provide@intraspace
% == Line breaking: justification ==
%\if\bbl@KVP@justification\@nnil\else
\let\bbl@KVP@linebreaking\bbl@KVP@justification
%\fi
%\if\bbl@KVP@linebreaking\@nnil\else
\bbl@xin{,\bbl@KVP@linebreaking}%
%\fi
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{%
  \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
  \@namedef{extras#1}{}%
  \@namedef{noextras#1}{}%
  \bbl@startcommands*{#1}{captions}%
  \ifx\bbl@KVP@captions\@nnil % and also if import, implicit
    \def\bbl@tempb##1{% elt for \bbl@captionslist
      \bbl@ifempty{\bbl@stripslash##1}{#1\bbl@stripslash##1}}%
      \expandafter\bbl@tempb\bbl@captionslist\@empty
    \else
      \ifx\bbl@initoload\relax
        \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
      \else
        \bbl@read@ini{\bbl@initoload}2% % Same
      \fi
      \bbl@startcommands{#1}%
      \ifx\bbl@KVP@date\@nnil
        \bbl@savetoday\bbl@savedate
      \else
        \bbl@read@ini{\bbl@initoload}2% % Same
      \fi
      \bbl@endcommands
  \fi
}
The hyphenrules option is handled with an auxiliary macro. This macro is called in three cases:
when a language is first declared with \babelprovide, with hyphenrules and with import.

\def\babel@provide@hyphens#1{%
  \@tempcnta\m@ne % a flag
  \ifax\babel@KVP@hyphenrules@\@nnil\else
    \bbx\babel@replace\bbx@KVP@Hyphenrules{ }{},% 
    \bbx@for\bbx@KVP@Hyphenrules{%
      \ifnum\@tempcnta\m@ne % if not yet found
        \bbx@ifsamestring{##1}{+}%
        \{\bbx@carg\addlanguage{l@##1}\}
      \}
      \bbx@ifunset{l@##1}% After a possible +
      \{
        \@tempcnta\nameuse{l@##1}\%
      \}
    \fi%
  \ifnum\@tempcnta\m@ne % if no opt or no language in opt found
    \ifx\bbx@KVP@captions@\@nnil % TODO. Hackish. See above.
      \bbx@ifunset{\bbx@hyphr@##1}% use value in ini, if exists
  \else
    \setbox\z@\hbox{\BabelBeforeIni{\bbx@initoload{}}}%
  \fi
  \}}

The hyphenrules option is handled with an auxiliary macro. This macro is called in three cases:
when a language is first declared with \babelprovide, with hyphenrules and with import.
The reader of babel...tex files. We reset temporarily some catcodes.

Now, the 'main loop', which **must be executed inside a group**. At this point, \bbl@inidata may contain data declared in \bbl@provide, 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.
A variant to be used when the ini file has been already loaded, because it's not the first \babelprovide for this language.
A somewhat hackish tool to handle calendar sections. TODO. To be improved.

A key with a slash in \babelprovide replaces the value in the ini file (which is ignored altogether).

The mechanism is simple (but suboptimal): add the data to the ini one (at this point the ini file has not yet been read), and define a dummy macro. When the ini file is read, just skip the corresponding key and reset the macro (in \bbl@inistore above).
The previous assignments are local, so we need to export them. If the value is empty, we can provide a default value.

```latex
\def\bbl@exportkey#1#2#3{\%\bbl@ifunset{bbl@@kv@#2}{\bbl@csarg\gdef{#1@languagename}{#3}\%\else\bbl@exp{\global\let<#1@languagename><bbl@@kv@#2}\%\fi}}
```

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

```latex
\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 }}\%\%\let\bbl@release@transforms@empty\let\bbl@release@casing@empty\def\bbl@ini@exports#1{\% Identification always exported\bbl@iniwarning{\%\ifcase\bbl@engine\bbl@iniwarning{.pdflatex}\or\bbl@iniwarning{.lualatex}\or\bbl@iniwarning{.xelatex}\or\bbl@iniwarning{.lualatex}\or\bbl@iniwarning{.xelatex}\fi\%\bbl@exportkey{llevel}{identification.load.level}\%\bbl@exportkey{elname}{identification.name.english}\%\bbl@exp{\bbl@exportkey{ lname}{identification.name.opentype}{\csname bbl@elname@languagename\endcsname}}\%\bbl@exportkey{tbcp}{identification.tag.bcp47}\%\% Somewhat hackish. TODO:\bbl@exportkey{casing}{identification.tag.bcp47}\%\bbl@exportkey{bcp}{identification.language.tag.bcp47}\%\bbl@exportkey{otf}{identification.script.tag.opentype}{DFLT}\%\bbl@exportkey{esname}{identification.script.name}\%\bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}{\csname bbl@esname@languagename\endcsname}}\%\bbl@exportkey{sbcp}{identification.script.tag.bcp47}\%\bbl@exportkey{bcp}{identification.script.tag.bcp47}\%\bbl@exportkey{cbcp}{identification.script.tag.bcp47}\%\bbl@exportkey{csoft}{identification.script.tag.opentype}{DFLT}\%\bbl@exportkey{rcbp}{identification.region.tag.bcp47}\%\bbl@exportkey{vcbp}{identification.variant.tag.bcp47}\%\bbl@exportkey{extt}{identification.extension.t.tag.bcp47}\%\bbl@exportkey{extu}{identification.extension.u.tag.bcp47}\%\bbl@exportkey{extx}{identification.extension.x.tag.bcp47}\%\% Also maps bcp47 -> languagename\bbl@bcptoname\%\bbl@csarg\xdef{bcp@map@\bbl@cl{tbcp}}{\languagename}\%\fi\%\bbl@release\%\directlua{\%\Babel.locale_props{\%the\bbl@cs{id@\languagename}.script = '\bbl@cl{sbcp}'}\%\}````
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'.
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.

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.

\def\bbl@list@the{%part, chapter, section, subsection, subsubsection, paragraph, %subparagraph, enumi, enumii, enumiii, enumiv, equation, figure, %table, page, footnote, mpfootnote, mpfn}\def\bbl@map@cnt#1{% #1: roman, etc, // #2: enumi, etc
\bbl@ifunset{bbl@map@#1@languagename}{{@nameuse{#1}}}{\@nameuse{bbl@map@#1@languagename}}}\def\bbl@inikv@labels#1#2{%\in@{.map}{#1}\ifin@\ifx\bbl@KVP@labels@nnil\else\bbl@xin@{ map }{ \bbl@KVP@labels}\ifin@\def\bbl@tempc{#1}\bbl@replace\bbl@tempc{.map}{\localecounter{#2}}{\bbl@map@cnt{#1}{#2}}\bbl@ifunset{the##1}{\bbl@exp{\let\bbl@tempd\the##1}\bbl@exp{\bbl@sreplace\the##1{\bbl@tempc}{##1}}{\bbl@map@cnt{#1}{#2}}}\expandafter\ifx\csname the##1\endcsname\bbl@tempd\else\toks@{\expandafter\expandafter\expandafter{\csname the##1\endcsname}}\bbl@exp{\bbl@add{\extras@languagename}{\babel@save{\labelenum\romannumeral\bbl@tempa}}{\def\labelenum\romannumeral\bbl@tempa{\the\toks@}}}\fi\fi\fi\fi\else%The following code is still under study. You can test it and make% suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's% language dependent.\in@{enumerate.}{#1}\ifin@\def\bbl@tempa{#1}\bbl@replace\bbl@tempa{enumerate.}{\nobreakspace{}}\bbl@replace\bbl@tempa{[}{\csname the}\bbl@replace\bbl@tempa{]}{\endcsname{}}\toks@{\expandafter\expandafter\expandafter{\csname the##1\endcsname{}}\bbl@exp{\bbl@add{\extras@languagename}{\bbl@tempa}{\def\bbl@tempa{\the\toks@}}}\fi}%The following code is still under study. You can test it and make% suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's% language dependent.\in@{enumerate.}{#1}\ifin@\def\bbl@tempa{#1}\bbl@replace\bbl@tempa{enumerate.}{\nobreakspace{}}\bbl@replace\bbl@tempa{[}{\csname the}\bbl@replace\bbl@tempa{]}{\endcsname{}}\toks@{\expandafter\expandafter\expandafter{\csname the##1\endcsname{}}\bbl@exp{\bbl@add{\extras@languagename}{\bbl@tempa}{\def\bbl@tempa{\the\toks@}}}\fi}
To show correctly some captions in a few languages, we need to patch some internal macros, because the order is hardcoded. For example, in Japanese the chapter number is surrounded by two strings, while in Hungarian is placed after. These replacement works in many classes, but not all. Actually, the following lines are somewhat tentative.

```latex
\def\bbl@chaptype{chapter}
\ifx\@makechapterhead\@undefined
  \let\bbl@patchchapter\relax
\else\ifx\thechapter\@undefined
  \let\bbl@patchchapter\relax
\else\ifx\ps@headings\@undefined
  \let\bbl@patchchapter\relax
\else
  \def\bbl@patchchapter{\
    \global\let\bbl@patchchapter\relax
    \gdef\bbl@chfmt{\
      \bbl@ifunset{bbl@\bbl@chaptype fmt@\languagename}\
      {\@chapapp\space\thechapter}
      {\@nameuse{bbl@\bbl@chaptype fmt@\languagename}}}
    \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}
    \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}
    \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}
  }
\fi\fi\fi
\fi\fi\fi

\let\bbl@patchappendix\bbl@patchchapter
\ifx\@part\@undefined
  \let\bbl@patchpart\relax
\else
  \def\bbl@patchpart{\
    \global\let\bbl@patchpart\relax
    \gdef\bbl@partformat{\
      \bbl@ifunset{bbl@partfmt@\languagename}{}{\bbl@cl{calpr}},\ #1
      {\@nameuse{bbl@partfmt@\languagename}}}
    \bbl@sreplace\@part{\partname\ 
    \bbl@patchappendix}
  }
\fi
```

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

```latex
\let\bbl@calendar\@empty
\DeclareRobustCommand\localedate[1][1]{\bbl@localedate{#1}}
\def\bbl@localedate#1#2#3#4{\
  \begingroup
    \edef\bbl@they{#2}\
    \edef\bbl@them{#3}\
    \edef\bbl@thed{#4}\
    \edef\bbl@tempe{\
      \bbl@ifunset{bbl@calpr@\languagename}{}{\bbl@cl{calpr}},#1
      {\@nameuse{bbl@calpr@\languagename}}}
    \bbl@replace\bbl@tempe{ }{}\
    \bbl@replace\bbl@tempe{CONVERT}{convert=}\% Hackish
  \endgroup
}\begin{group}
  \edef\bbl@they{#2}\
  \edef\bbl@them{#3}\
  \edef\bbl@thed{#4}\
  \edef\bbl@tempe{\
    \bbl@ifunset{bbl@calpr@\languagename}{}{\bbl@cl{calpr}},#1
    \bbl@replace\bbl@tempe{ })\}%
  \bbl@replace\bbl@tempe{CONVERT}{convert=}\% Hackish
\end{group}
```

69
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 either in the date or in the month name. Note after `\bbl@replace` `\toks@` contains the resulting string, which is used by `\bbl@replace@finish@iii` (this implicit behavior doesn't seem a good idea, but it's efficient).
\newcommand\BabelDateMMMM[1]{% 
\csname month\romannumeral#1\bbl@calendar name\endcsname}} 
\newcommand\BabelDatey[1]{\number#1} 
\newcommand\BabelDateyy[1]{% 
\ifnum#1<10 \number#1 \else \ifnum#1<100 \number#1 \else \ifnum#1<1000 \expandafter\@gobble\number#1 \else \ifnum#1<10000 \expandafter\@gobbletwo\number#1 \else \bbl@error{limit-two-digits}{}{}{}\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{[|}{\bbl@datecntr[####1|}}% 
\bbl@replace\bbl@toreplace{[|}{\bbl@datecntr[####1|}}% 
\bbl@replace\bbl@toreplace{[|}{\bbl@datecntr[####2|}}% 
\bbl@replace\bbl@toreplace{[|}{\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.

\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+", '')
token.set_macro('babeltempa', str)
}&% 
\def\babeltempc{}&% 
\bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&% 
\ifin@ &% font:font:transform syntax
\directlua{ 
local t = {}
for m in string.gsub('#1', '([^\n]+)', '\bbl@xdatecntr\#1\#2\}'
do table.insert(t, m)
end

71
Language and Script values to be used when defining a font or setting the direction are set with the following macros.

```latex
\def\bbl@provide@lsys#1{\% \def\bbl@ifunset#1{, #1}}\def\bbl@exp#1{\relax \ifblank{#1}{}{\ifx\AtBeginDocument\@notprerr\fi \AtBeginDocument{\expandafter\select@language\expandafter{\languagename}}} \ifcase\bbl@engine\or\or \bbl@csarg\let\lsys@#1@empty \bbl@csarg\gdef\sname@#1{Default} \bbl@csarg\gdef\sotf@#1{DFLT} \bbl@csarg\add@list{\lsys@#1}{\script=*\sname@#1} \if\bbl@lname@#1\bbl@csarg\add@list{\lsys@#1}{\language=*\bbl@cs;lname@#1} \expandafter\@secondoftwo \fi \global\let\bbl@xenohyph\bbl@xenohyph@d \expandafter\select@language\expandafter{\languagename} \fi}} \bbl@csarg\bbl@toglobal{\lsys@#1}}
```

Language and Script values to be used when defining a font or setting the direction are set with the following macros.
\bb@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
\fi}%
\{\hyphenchar\font\defaulthyphenchar%
\}%

The following ini reader ignores everything but the identification section. It is called when a font is defined (ie, when the language is first selected) to know which script/language must be enabled. This means we must make sure a few characters are not active. The ini is not read directly, but with a proxy file named as the language (which means any code in it must be skipped, too).
\def\bbl@load@info#1{%
\def\BabelBeforeIni##1##2{%
\begingroup
\bbl@read@ini{##1}0%
\endinput % babel- .tex may contain only preamble's
\endgroup}{\bbl@input@texini{#1}}}

A tool to define the macros for native digits from the list provided in the ini file. Somewhat convoluted because there are 10 digits, but only 9 arguments in \TeX. Non-digits characters are kept. The first macro is the generic "localized" command.
\def\bbl@setdigits#1#2#3#4#5{%
\bbl@exp{%\def\languagename digits####1{% ie, \langdigits
\let\bbl@digits@languagename####1\@nil}%
\def\languagename counter####1{% ie, \langcounter
\expandafter\bbl@counter@languagename%\csname c####1\endcsname}%
\def\bbl@counter@languagename####1{% ie, \bbl@counter@lang
\expandafter\bbl@digits@languagename\number####1\@nil}}%
\def\bbl@tempa##1##2##3##4##5{%\bbl@exp{\def\bbl@digits@languagename########1{\ifx########1\@nil % ie, \bbl@digits@lang
\else\ifx0########1#1\
\else\ifx1########1#2\
\else\ifx2########1#3\
\else\ifx3########1#4\
\else\ifx4########1#5\
\else\ifx5########1#6\
\else\ifx6########1#7\
\else\ifx7########1#8\
\else\ifx8########1#9\
\else########1
\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
\expandafter\bbl@digits@languagename%\fi}}%
\bbl@tempa}

Alphabetic counters must be converted from a space separated list to an \ifcase structure.
\def\bbl@buildifcase#1 {\% Returns \bbl@tempa, requires \toks=0\}
\def\ifx\#1\% \% before, in case #1 is multiletter
\bbl@exp(%

73
The code for additive counters is somewhat tricky and it's based on the fact the arguments just before \@ collects digits which have been left 'unused' in previous arguments, the first of them being the number of digits in the number to be converted. This explains the reverse set 76543210. Digits above 10000 are not handled yet. When the key contains the subkey \textit{.F.}, the number after is treated as a special case, for a fixed form (see babel-he.ini, for example).

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

The information in the identification section can be useful, so the following macro just exposes it with a user command.
LA TEX needs to know the BCP 47 codes for some features. For that, it expects BCP data to be defined. While language, region, script, and variant are recognized, extension ⟨⟩ 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
% Still somewhat hackish. WIP. Note |\str_if_eq:nnTF| is fully
% expandable (|\bbl@ifsamestring| isn't).
\providecommand\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-ini-field}{#1}{}{}}%
  {\bbl@ifunset{bbl@csname bbl@info@#1.tag.bcp47@#2}{}%
    {\bbl@cs{bbl@info@#1.tag.bcp47@#2}}}}%
\fi
\namedef{bbl@info@casing.tag.bcp47}{casing}
\newcommand\BabelUppercaseMapping[3]{%
  DeclareUppercaseMapping[|\nameuse{bbl@casing@#1}|]{#2}{#3}}%
\newcommand\BabelTitlecaseMapping[3]{%
  DeclareTitlecaseMapping[|\nameuse{bbl@casing@#1}|]{#2}{#3}}%
\newcommand\BabelLowercaseMapping[3]{%
  DeclareLowercaseMapping[|\nameuse{bbl@casing@#1}|]{#2}{#3}}%
The parser for casing and casing. ⟨variant⟩.
\begin{verbatim}
\def\bbl@casemapping#1#2#3{| 1:variant
  \def\bbl@tempa##1 ##2{%
    Loop
    \bbl@casemapping@i{##1}%
    \ifx\@empty##2\else\bbl@afterfi\bbl@tempa##2\fi}%
  \edef\bbl@templ{\@nameuse{bbl@casing@#2}#1}% Language code
  \def\bbl@tempe{0}% Mode (upper/lower...)
  \def\bbl@tempc{#3 }% Casing list
  \expandafter\bbl@casemapping@ii\bbl@tempc\@empty}
\def\bbl@casemapping@i#1{% 1: Loop
  \bbl@casemapping@ii[#1]}
\def\bbl@casemapping@ii[1]{%
  \ifcase\bbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
    \@nameuse{regex_replace_all:nnN}{[\x{c0}-\x{ff}][\x{80}-\x{bf}]*}{\0}\bbl@tempb%
  \else
    \@nameuse{regex_replace_all:nnN}{.}{\0}
  \fi}
\def\bbl@casemapping@i[#1]{% 2: Casing list
  \expandafter\bbl@casemapping@ii[\bbl@tempe]\bbl@tempc\@empty}
\end{verbatim}
\end{verbatim}

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

\let\bbl@ensureinfo\@gobble
\newcommand\BabelEnsureInfo{\ifx\InputIfFileExists\@undefined\else
\def\bbl@ensureinfo##1{\bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}}\fi
\bbl@foreach\bbl@loaded{{\let\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files
\def\languagename{##1}\bbl@ensureinfo{##1}}}
\@ifpackagewith{babel}{ensureinfo=off}{}% {\AtEndOfPackage{% Test for plain.
\ifx\@undefined\bbl@loaded\else\BabelEnsureInfo\fi}

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

\newcommand\getlocaleproperty{%
\@ifstar\bbl@getproperty@s\bbl@getproperty@x}
\def\bbl@getproperty@s#1#2#3{\let#1\relax\def\bbl@elt##1##2##3{\bbl@ifsamestring{##1/##2}{#3}\providecommand#1{##3}\def\bbl@elt####1####2####3{}}}\bbl@cs{inidata@#2}}\def\bbl@getproperty@x#1#2#3{\bbl@getproperty@s{#1}{#2}{#3}\ifx#1\relax\bbl@error{unknown-locale-key}{#1}{#2}{#3}\fi}
\let\bbl@ini@loaded\@empty
\newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
\def\ShowLocaleProperties#1{\typeout{}\typeout{*** Properties for language '#1' ***}\def\bbl@elt##1##2##3{\typeout{##1/##2 = ##3}}\@nameuse{bbl@inidata@#1}\typeout{*******}}

5 Adjusting the Babel behavior
A generic high level interface is provided to adjust some global and general settings.

\newcommand\babeladjust[1]{% TODO. Error handling.
\bbl@forkv{#1}{\bbl@ifunset{bbl@ADJ@##1@##2}{\bbl@error{unknown-adjust-key}{##1}{##2}{##3}}}}
\def\bbl@adjust@lua#1#2{\ifvmode\ifnum\currentgrouplevel=\z@\directlua{ Babel.#2 }\expandafter\expandafter\expandafter\@gobble\fi\fi}{\bbl@error{adjust-only-vertical}{#1}{}{}}% Gobbled if everything went ok.
@namedef{bbl@ADJ@bidi.mirroring@on}{\bbl@adjust@lua{bidi}{mirroring_enabled=true}}
@namedef{bbl@ADJ@bidi.mirroring@off}{\bbl@adjust@lua{bidi}{mirroring_enabled=false}}
@namedef{bbl@ADJ@bidi.text@on}{\bbl@adjust@lua{bidi}{bidi_enabled=true}}
@namedef{bbl@ADJ@bidi.text@off}{\bbl@adjust@lua{bidi}{bidi_enabled=false}}
@namedef{bbl@ADJ@bidi.math@on}{\let\bbl@noamsmath\@empty}
@namedef{bbl@ADJ@bidi.math@off}{\let\bbl@noamsmath\relax}
@namedef{bbl@ADJ@bidi.mapdigits@on}{\bbl@adjust@lua{bidi}{digits_mapped=true}}
@namedef{bbl@ADJ@bidi.mapdigits@off}{\bbl@adjust@lua{bidi}{digits_mapped=false}}
@namedef{bbl@ADJ@linebreak.sea@on}{\bbl@adjust@lua{linebreak}{sea_enabled=true}}
@namedef{bbl@ADJ@linebreak.sea@off}{\bbl@adjust@lua{linebreak}{sea_enabled=false}}
@namedef{bbl@ADJ@linebreak.cjk@on}{\bbl@adjust@lua{linebreak}{cjk_enabled=true}}
@namedef{bbl@ADJ@linebreak.cjk@off}{\bbl@adjust@lua{linebreak}{cjk_enabled=false}}
@namedef{bbl@ADJ@justify.arabic@on}{\bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
@namedef{bbl@ADJ@justify.arabic@off}{\bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
\def\bbl@adjust@layout#1{\ifvmode#1\expandafter\@gobble\fi}{\bbl@error{layout-only-vertical}{}{}{}}% Gobbled if everything went ok.
@namedef{bbl@ADJ@layout.tabular@on}{\ifnum\bbl@tabular@mode=\tw@bl@adjust@layout{\let\tabular\bbl@NL@@tabular}\else\chardef\bbl@tabular@mode\@ne\fi}
@namedef{bbl@ADJ@layout.tabular@off}{\ifnum\bbl@tabular@mode=\tw@bl@adjust@layout{\let\tabular\bbl@OL@@tabular}\else\charge{\bbl@tabular@mode\@ne}\fi}
@namedef{bbl@ADJ@layout.lists@on}{\bbl@adjust@layout{\let\list\bbl@NL@list}}
@namedef{bbl@ADJ@layout.lists@off}{\bbl@adjust@layout{\let\list\bbl@OL@list}}
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.

3686 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3687 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3688 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3689 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3690 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}

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

3691 \bbl@trace{Cross referencing macros}
3692 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3693 \edef\@multiplelabels{\@latex@warning@no@line{There were multiply-defined labels}}%
3694 \@latex@warning@no@line{Label `#2' multiply defined}%
3695 \global\@namedef{#1@#2}{#3}
3696 \fi

\testdef 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.

3703 \CheckCommand*\testdef[3]{%
3704 \def\reserved@a{#3}%
3705 \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname\reserved@a
3706 \else
3707 \@tempswatrue
3708 \fi

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

3709 \def\testdef#1#2#3{% TODO. With \@samestring?
3710 \@safe@activestrue
3711 \expandafter\let\expandafter\bbl@tempa\csname #1#2\endcsname
3712 \def\bbl@tempb#3{%
3713 \@safe@activesfalse
3714 \ifx\bbl@tempa\@relax
3715 \else
3716 \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3717 \fi
3718 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3719 \ifx\bbl@tempa\bbl@tempb
3720 \else
3721 \@tempswatrue
3722 \fi
3723 \fi

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

3724 \bbl@xin{R}\bbl@opt@safe
3725 \fi}
3726 \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3727 \bbl@xin{\expandafter\strip@prefix\meaning\bbl@tempc}%
3728 \expandafter\strip@prefix\meaning\ref%
3729 \ifi
\@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.

\AtBeginDocument{%
\@ifpackageloaded{natbib}{%
\def\@citex[#1][#2]{%\@safe@activestrue\edef\bbl@tempa{#2}\@safe@activesfalse
\org@@citex[#1][#2]{\bbl@tempa}}%
}}%

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

\AtBeginDocument{%
\@ifpackageloaded{cite}{%
\def\@citex[#1]{%\@safe@activestrue\org@@citex[#1]{\@safe@activesfalse}
}}%

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

\AtBeginDocument{%
\@ifpackageloaded{natbib}{%
\def\cite[#1][#2][#3]{%\@safe@activestrue\edef\bbl@tempa[#3]{\@safe@activesfalse
\org@\cite[#1][#2]{\bbl@tempa}}}
}}%

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

\AtBeginDocument{%
\@ifpackageloaded{cite}{%
\def\cite[#1][#2]{%\@safe@activestrue\org@\cite[#1][#2]{\@safe@activesfalse}
}}%

\@citex

\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@cite@choice

The macro \bbl@cite@choice 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.

5.2 Marks

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.
The definition of \markboth is equivalent to that of \markright, except that we need two 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.)

\begin{verbatim}
3806  \ifx\@mkboth\markboth
3807   \def\bbl@tempc{\let\@mkboth\markboth}%
3808  \else
3809   \def\bbl@tempc{}%
3810  \fi
3811  \bbl@ifunset{markboth }\bbl@redefine\bbl@redefinerobust
3812  \markboth#1#2{%
3813   \protected@edef\bbl@tempb##1{%
3814      \protect\foreignlanguage{\languagename}{\protect\bbl@restore@actives##1}}%
3815   \bbl@ifblank{#1}{}%
3816   {\toks@{}%  
3817      {\toks@\expandafter{\bbl@tempb{#1}}}%  
3818      {\bbl@ifblank{#2}{}%
3819         {\@temptokena{}%
3820            {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3821            {\bbl@exp{\org@markboth{\the\toks@}{\the\@temptokena}}}%
3822            \bbl@tempc%
3823            \fi} % end ifbbl@single, end \IfBabelLayout
\end{verbatim}

5.3 Preventing clashes with other packages

5.3.1 ifthen

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

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.

\begin{verbatim}
3825  \bbl@trace{Preventing clashes with other packages}
3826  \ifx\org@ref\@undefined\else
3827   \bbl@xin@{R}\bbl@opt@safe
3828  \ifin@
3829   \AtBeginDocument{%
3830     \@ifpackageloaded{ifthen}{%
3831        \bbl@redefine\long\ifthenelse{1#2#3}{%
3832           \let\bbl@temp@pref\pageref
3833           {\let\pageref\org@pageref
3834            \let\ref\org@ref
3835           \@safe@activestruen
3836            {\org@ifthenelse{1}%
3837                {\let\pageref\bbl@temp@pref
3838                 \let\ref\bbl@temp@ref
3839               \@safe@activesfalse
3840               \@ifthenelse{2}%

82
\end{verbatim}
5.3.2 varioref

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

```
\AtBeginDocument{%
  \@ifpackageloaded{varioref}{%
    \bb@redefine\@@vpageref#1[#2]#3{%\@safe@activestrue\org@@@vpageref{#1}[#2]{#3}%;\@safe@activesfalse}%;
    \bb@redefine\vrefpagenum#1#2{%\@safe@activestrue\org@vrefpagenum{#1}{#2}%;\@safe@activesfalse}%
  }%
}
```

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

```
\expandafter\def\csname Ref \endcsname#1{%\protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
```

5.3.3 hhline

```
\hhline
```

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

```
\AtEndOfPackage{%
  \AtBeginDocument{%
    \@ifpackageloaded{hhline} {
      \expandafter\ifx\csname normal@char\string: \endcsname\relax
        \else
          \makeatletter
          \def@currcode{hhline}\input{hhline.sty}\makeatother
        \fi}
    \}
  }
}
```

\substitutefontfamily

*Deprecated.* Use the tools provided by EPiX. 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.

```
\def\substitutefontfamily#1#2#3{%\lowercase{\immediate\openout15=#1#2.fd}\relax%\immediate\write15{\string\ProvidesFile{#1#2.fd}[	he\year/\two@digits{\the\month}/\two@digits{\the\day}\space generated font description file]^^J}
```

83
5.4 Encoding and fonts

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

\ensureascii{}

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 latinencoding to use.
end of processing the package is the Latin encoding.

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

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

\AtBeginDocument{%
\ifpackageloaded{fontspec}%
  \xdef\latinencoding{%
    \ifx\UTFencname\@undefined
      EU\ifcase\bbl@engine\or2\or1\fi
    \else
      \UTFencname
    \fi}
  \else
    \UTFencname
  \fi%
}{
  \gdef\latinencoding{OT1}%
  \ifx\cf@encoding\bbl@t@one
    \xdef\latinencoding{\bbl@t@one}
  \else
    \def\@elt#1{,#1,}
    \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}
    \let\@elt\relax
    \bbl@xin{,T1,}\bbl@tempa
    \ifin@
    \xdef\latinencoding{\bbl@t@one}
    \fi
  \fi}
%
\latintext

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

\DeclareRobustCommand{\latintext}{%
  \fontencoding{\latinencoding}\selectfont
  \def\encodingdefault{\latinencoding}}
%
\textlatin

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

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

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

\iffalse\bbl@patchfont#1{\AddToHook{selectfont}{#1}}

\fi

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 rl babel.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 \texttt{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 \texttt{lists}, \texttt{columns}, \texttt{counters}, \texttt{toc}s, much like \texttt{rl 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 \TeX grouping.

• luatex can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but bidi text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As Lua\TeX ja shows, vertical typesetting is possible, too.

\begin{verbatim}
\inputencoding{utf8x}\usepackage{babel}\usepackage{bidi}
\begin{document}
Now come the macros used to set the direction when a language is switched. First the (mostly) common macros.
\end{verbatim}
Now the engine-dependent macros. TODO. Must be moved to the engine files.

\ifodd\bbl@engine % luatex=1
  \else \% pdftex=0, xetex=2
  \newcount\bbl@dirlevel
  \chardef\bbl@thetextdir\z@
  \chardef\bbl@thepardir\z@
  \def\bbl@textdir#1{% 
    \ifcase#1 \relax
      \chardef\bbl@thetextdir\z@
      \@nameuse{setlatin} % \bbl@textdir@i\beginL\endL
    \else
      \chardef\bbl@thetextdir\@ne
      \@nameuse{setnonlatin} % \bbl@textdir@i\beginR\endR
    \fi
  }% \bbl@textdir@i#1#2% 
  \def\bbl@textdir@i{% 
    \ifhmode
      \ifnum\currentgrouplevel>\z@
        \ifnum\currentgrouplevel=\bbl@dirlevel
          \bbl@error{multiple-bidi}{}{}{}% 
        \bgroup\aftergroup#2\aftergroup\egroup
        \else
          \ifcase\currentgrouptype
            \aftergroup#2 % 1 simple {}
          \or
            \aftergroup\endgroup% 0 bottom
          \fi
        \fi
      \fi
    \fi
  }% \bbl@textdir@i#1#2%
  \def\bbl@textdir@i#1{% 
    \ifnum\currentgrouplevel>\z@
      \ifnum\currentgrouplevel=\bbl@dirlevel
        \bbl@error{multiple-bidi}{}{}{}
      \else
        \ifcase\currentgrouptype
          \aftergroup\endgroup% 0 bottom
        \or
          \aftergroup\endgroup% 1 simple {}
        \fi
      \fi
    \fi
  }% \bbl@textdir@i#1#2%
  \def\bbl@textdir@i#1#2%
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 L (mainly digits). We also disable warnings with hyperref.

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

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

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

4126\bbl@trace{Local Language Configuration}
4127\ifx\loadlocalcfg\@undefined
4128\@ifpackagewith{babel}{noconfigs}\
4129\{\let\loadlocalcfg\@gobble\%
4130\{\def\loadlocalcfg#1{\%
4131\InputIfFileExists{#1.cfg}\%
4132\{\typeout{*************************************^^J%
4133\* Local config file #1.cfg used^^J%
4134\*}}\%
4135\@empty\%
4136\fi

5.7 Language options

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

4137\bbl@trace{Language options}
4138\let\bbl@afterlang\relax
4139\let\BabelModifiers\relax
4140\let\bbl@loaded\@empty
4141\def\bbl@load@language#1{\%
4142\InputIfFileExists{#1.ldf}\%
4143\{\edef\bbl@loaded{\CurrentOption\%
4144\ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}\%
4145\expandafter\let\expandafter\bbl@afterlang\%
4146\csname\CurrentOption.ldf-@@k\endcsname\%
4147\expandafter\let\expandafter\BabelModifiers\%
4148\csname bbl@mod@\CurrentOption\endcsname\%
4149\bbl@exp{\\AtBeginDocument{\\bbl@usehooks@lang{\CurrentOption}{begindocument}{\CurrentOption}}}\%
4150\IfFileExists{babel-#1.tex}{\def\bbl@tempa{\%
4151\There is a locale ini file for this language.\%
4152\If it’s the main language, try adding ‘provide=’\%
4153\to the babel package options)}\%
4154\{\let\bbl@tempa\empty\%
4155\bbl@error{unknown-package-option}{}{}{}}\%
4156\}
4157\bbl@try@load@lang#1#2#3{\%
4158\IfFileExists{#1.ldf}{\bbl@load@language{#1}}{\bbl@load@language{#2}#3}}

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.

4159\DeclareOption{hebrew}{\%
4160\Ifcase\bbl@engine\or
4161\bbl@error{only-pdftex-lang}{hebrew}{\latex}\%
4162\fi\%
4163\input{rlbabel.def}\%
4164\bbl@load@language{hebrew}}\%
4165\DeclareOption{hungarian}{\bbl@try@load@lang\{}{magyar}\{}}\%
4166\DeclareOption{lowersorbian}{\bbl@try@load@lang\{}{lsorbian}\{}}\%
4167\DeclareOption{polutonikogreek}{\bbl@try@load@lang\{}{polon}\{}}\%
4168\DeclareOption{polon}\%
4169\DeclareOption{polon}\%
4170\pagebreak
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, 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.

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):

```verbatim
\def\AfterBabelLanguage#1{% 
  \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
\DeclareOption*{}
\ProcessOptions*
```

This finished the second pass. Now the third one begins, which loads the main language set with the key `main`. A warning is raised if the main language is not the same as the last named one, or if the value of the key main is not a language. With some options in provide, the package luatexbase is loaded (and immediately used), and therefore \bbl@provide can't go inside a \DeclareOption; this explains why it's executed directly, with a dummy declaration. Then all languages have been loaded, so we deactivate \AfterBabelLanguage.

```verbatim
\def\AfterBabelLanguage#1{% 
  \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
\DeclareOption*{}
\ProcessOptions*
```

This finished the second pass. Now the third one begins, which loads the main language set with the key `main`. A warning is raised if the main language is not the same as the last named one, or if the value of the key main is not a language. With some options in provide, the package luatexbase is loaded (and immediately used), and therefore \bbl@provide can't go inside a \DeclareOption; this explains why it's executed directly, with a dummy declaration. Then all languages have been loaded, so we deactivate \AfterBabelLanguage.

```verbatim
\def\AfterBabelLanguage#1{% 
  \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
\DeclareOption*{}
\ProcessOptions*
```

This finished the second pass. Now the third one begins, which loads the main language set with the key `main`. A warning is raised if the main language is not the same as the last named one, or if the value of the key main is not a language. With some options in provide, the package luatexbase is loaded (and immediately used), and therefore \bbl@provide can't go inside a \DeclareOption; this explains why it's executed directly, with a dummy declaration. Then all languages have been loaded, so we deactivate \AfterBabelLanguage.

```verbatim
\def\AfterBabelLanguage#1{% 
  \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
\DeclareOption*{}
\ProcessOptions*
```

This finished the second pass. Now the third one begins, which loads the main language set with the key `main`. A warning is raised if the main language is not the same as the last named one, or if the value of the key main is not a language. With some options in provide, the package luatexbase is loaded (and immediately used), and therefore \bbl@provide can't go inside a \DeclareOption; this explains why it's executed directly, with a dummy declaration. Then all languages have been loaded, so we deactivate \AfterBabelLanguage.
6 The kernel of Babel (babel.def, common)

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

Because plain \TeX users might want to use some of the features of the babel system too, care has to be
taken that plain \TeX can process the files. For this reason the current format will have to be checked
in a number of places. Some of the code below is common to plain \TeX and \LaTeX, some of it is for the
\LaTeX case only.

Plain formats based on etex (etex, etex1, luatex) don't load hyphen.cfg but etex.src, which follows
a different naming convention, so we need to define the babel names. It presumes language.def
exists and it is the same file used when formats were created.

A proxy file for switch.def

\let\bbl@onlyswitch\empty
\input babel.def
\let\bbl@onlyswitch\undefined

% They are loaded when \bbl@error is first called. To save space, the
% main code just identifies them with a tag, and messages are stored in
% a separate file. Since it can be loaded anywhere, you make sure some
% catcodes have the right value, although those for |\|, |`|, |^[M|, |%
% and |=* are reset before loading the file.
I'll insert a new group, but expect wrong results.
\bbl@errmessage{unknown-package-option}
\bbl@errmessage{unknown-package-option}
{Unknown option '\CurrentOption'. Either you misspelled it\%
 or the language definition file \CurrentOption.ldf\%
 was not found%
 \bbl@tempa} Valid options are, among others: shorthands=, KeepShorthandsActive,\
 activeacute, activegrave, noconfigs, safe=, main=, math=\%
 headfoot=, strings=, config=, hyphenmap=, or a language name.}
\bbl@errmessage{config-not-found} Local config file '\bbl@opt@config.cfg' not found%
\bbl@errmessage{late-after-babel} Too late for \string\AfterBabelLanguage%
\bbl@errmessage{double-hyphens-class} Double hyphens aren't allowed in \string\babelcharclass\%
 because it's potentially ambiguous)%
\bbl@errmessage{unknown-interchar} '%#1' for '\languagename' cannot be enabled.\%
 Maybe there is a typo.)%\bbl@errmessage{unknown-interchar-b} '%#1' for '\languagename' cannot be disabled.\%
 Maybe there is a typo.)%\bbl@errmessage{charproperty-only-vertical} \string\babelcharproperty\space can be used only in\%
 vertical mode (preamble or between paragraphs))%\bbl@errmessage{unknown-char-property} \bbl@errmessage{unknown-char-property} No property named '#2'. Allowed values are\%
 direction (bc), mirror (bm), and linebreak (lb)%\bbl@errmessage{bad-transform-option} \bbl@errmessage{bad-transform-option} Bad option '#1' in a transform.\%
 I'll ignore it but expect more errors)%\bbl@errmessage{font-conflict-transforms} \bbl@errmessage{font-conflict-transforms} Transforms cannot be re-assigned to different\%
 fonts. The conflict is in '\bbl@kv@label'.\%
 Apply the same fonts or use a different label)%\bbl@errmessage{transform-not-available} \bbl@errmessage{transform-not-available} '%#1' for '\languagename' cannot be enabled.\%
 Maybe there is a typo or it's a font-dependent transform)%\bbl@errmessage{transform-not-available-b} \bbl@errmessage{transform-not-available-b} '%#1' for '\languagename' cannot be disabled.\%
 Maybe there is a typo or it's a font-dependent transform)%
\bbl@errmessage{year-out-range} {Year out of range.\%
 The allowed range is #1}%\bbl@errmessage{only-pdfTeX-lang} {The '#1' ldf style doesn't work with #2,\%
 but you can use the ini locale instead.\%
 Try adding 'provide=' to the option list. You may\%
 also want to set 'bidi=' to some value.}%\bbl@errmessage{only-pdfTeX-lang} {The manual for further details.}
7 Loading hyphenation patterns

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

\def\ProvidesFile{hyphen.cfg}{
\langle\langle date \rangle\rangle v\langle\langle version \rangle\rangle Babel hyphens}
\xdef\bbl@format{\jobname}
def\bbl@version{\langle\langle version \rangle\rangle}
def\bbl@date{\langle\langle date \rangle\rangle}
def\@empty{}
\def\bbl@languages{}
\toks@{}\def\bbl@languages{}
\def\process@language#1#2#3{\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}
def\process@line\process@line
Each line in the file \texttt{language.dat} is processed by \texttt{\process@line} after it is read. The first thing this macro does is to check whether the line starts with \texttt{=}. When the first token of a line is an \texttt{=}, the macro \texttt{\process@synonym} is called; otherwise the macro \texttt{\process@language} will continue.

\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}

\process@language
The macro \texttt{\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 \texttt{\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 \texttt{language.dat} by adding for instance ‘:T1’ to the name of the language. The macro \texttt{\bbl@get@enc} extracts the font encoding from the language name and stores it in \texttt{\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{en}` and `\uccode{en}` 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.

```latex
\bbl@get@enc
\bbl@hyph@enc
```

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

```latex
\bbl@get@enc\bbl@hyph@enc
```

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. `loadkernel` currently loads nothing, but define some basic macros instead.

```latex
\bbl@get@everylanguage#1{}
\bbl@get@loadpatterns#1\input #1\relax
\let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
```

97
\readconfigfile  \ The configuration file can now be opened for reading.

4634 \openin1 = language.dat
See if the file exists, if not, use the default hyphenation file hyphen.tex. The user will be informed about this.

4635 \def\ languagename{english}%
4636 \ifeof1
4637 \message{I couldn’t find the file language.dat, I will try the file hyphen.tex}
4638 \input hyphen.tex\relax
4639 \chardef\lenglish@z@
4641 \else

Pattern registers are allocated using count register \last@language. Its initial value is 0. The definition of the macro \newlanguage is such that it first increments the count register and then defines the language. In order to have the first patterns loaded in pattern register number 0 we initialize \last@language with the value \-1.

4642 \ \last@language\m@ne

We now read lines from the file until the end is found. While reading from the input, it is useful to switch off recognition of the end-of-line character. This saves us stripping off spaces from the contents of the control sequence.

4643 \loop
4644 \endlinechar\m@ne
4645 \read1 to \bbl@line
4646 \endlinechar`\^^M
If the file has reached its end, exit from the loop here. If not, empty lines are skipped. Add 3 space characters to the end of \bbl@line. This is needed to be able to recognize the arguments of \process@line later on. The default language should be the very first one.

4647 \if T\ifeof1F\fi T\relax
4648 \ifx\bbl@line\@empty\else
4649 \edef\bbl@line{\bbl@line\space\space\space}\
4650 \expandafter\process@line\bbl@line\relax
4651 \fi
4652 \repeat
Check for the end of the file. We must reverse the test for \ifeof without \else. Then reactivate the default patterns, and close the configuration file.

4653 \begingroup
4654 \def\bbl@elt#1#2#3#4{%
4655 \global\language=#2\relax
4656 \global\languagename[1]%
4657 \def\bbl@elt##1##2##3##4{}%}
4658 \bbl@languages
4659 \endgroup
4660 \fi
4661 \closein1

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

4662 \if/\the\toks@\else
4663 \errhelp{language.dat loads no language, only synonyms}
4664 \errmessage{Orphan language synonym}
4665 \fi
Also remove some macros from memory and raise an error if \toks@ is not empty. Finally load switch.def, but the latter is not required and the line inputting it may be commented out.

4666 \let\bbl@line\undefined
4667 \let\process@line\undefined
4668 \let\process@synonym\undefined
4669 \let\process@language\undefined
4670 \let\bbl@get@enc\undefined
4671 \let\bbl@hyph@enc\undefined
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].

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.

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.
If the family in the previous command does not exist, it must be defined. Here is how:

\def\bbl@providefam#1{% 
\expandafter\def\expandafter\bbl@providefam\expandafter{\expandafter{#1}}% Just define it 
\newcommand{#1default}{}% Just define it 
\bbl@add\list\bbl@font@fams{#1}% 
\DeclareRobustCommand{#1family}{% \not@math@alphabet#1relax% % \prepare@family@series@update{#1}{#1default}% TODO. Fails 
\fontfamily{#1default}% \ifx>\UseHooks\@undefined\else\UseHook{#1family}\fi% 
\selectfont}% 
\DeclareTextFontCommand{#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{% 
\ifunset{bbl@WFF@f@family}{\gdef{WFF@f@family}{}% Flag, to avoid dupl warns 
\info@warn{The current font is not a babel standard family:\#1% 
\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\bbl@font.\%
See the manual for further details about \string\bbl@font.\%
Reported}}% 
\global\let\bbl@tempa{}% TODO. Don't use \bbl@tempa 
\foreach\bbl@font@fams{% don't gather with prev for 
\ifunset{bbl@##1dflt@}{\cs{famrst@##1}{\global\csarg\let{famrst@##1}\relax}{}% order is relevant. TODO: but sometimes wrong! 
\bbl@font@rst{#1}{#1family}{#1}{#1default}{#1family}{#1}}% the main part!
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\bbl@tempa\@empty
        \bbl@foreach\bbl@font@fams{%\bbl@ifunset{bbl@##1dflt@}{\@nameuse{##1family}{\bbl@csarg\gdef{WFF@\f@family}{}% Flag\bbl@exp{\bbl@add{\bbl@tempa{* \space\fontname\font}}}%\bbl@exp{%\xdef{##1dflt@}{\f@family}}}%\expandafter\xdef\csname##1default\endcsname{\f@family}}%\ifx\bbl@tempa\@empty\else\bbl@infowarn{The following font families will use the default settings for all or some languages:\bbl@tempa There is nothing intrinsically wrong with it, but\\ 'babel' will no set Script and Language, which could\\ be relevant in some languages. If your document uses\\ these families, consider redefining them with \string\babelfont.\\ Reported}\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 \texttt{ssub*}).

\begin{verbatim}
\def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
  \bbl@xin{<>}{#1}\
  \ifin@
    \bbl@exp{\bbl@fontspec@set\#1\expandafter\@gobbletwo#1\#3}\
  \fi
  \bbl@exp{\def\#2{\bbl@rmdflt@lang}}\bbl@ifsamestring{\f@series}{\bfdefault}{\bfseries}{}\let\bbl@tempa\relax}
\end{verbatim}

\begin{verbatim}
\def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
  \let\bbl@tempe\bbl@mapselect\edef\bbl@tempb{\bbl@stripslash#4/}% Catcodes hack (better pass it).
  \bbl@exp{\bbl@replace\bbl@tempb{\bbl@stripslash\family/}}%}
\end{verbatim}

102
\let\bbl@mapselect\relax
\let\bbl@temp@fam#4% eg, \texttt{\textbackslash rmfamily}, to be restored below
\let#4\@empty % Make sure \renewfontfamily is valid
\bbl@exp{%
\let\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, \texttt{\textbackslash rmfamily \textbackslash}
\keys_if_exist:nnF{fontspec-opentype}{Script/\bbl@cl{sname}}%\bbl@cl{sotf}\}
\keys_if_exist:nnF{fontspec-opentype}{Language/\bbl@cl{lname}}%\bbl@cl{lotf}\}
\let\bbl@tempfs@nx\<__fontspec_warning:nx>%\bbl@tempfs@nxx\let\<__fontspec_warning:nxx>\bbl@fs@warn@nx
\renewfontfamily\#4%
\[bl@cl{lsys},%
\ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
#2\}]{#3}% ie \bbl@exp{..}{#3}
\bbl@exp{%\let\bbl@temp@fam\bbl@temp@pfam\let\bbl@mapselect\bbl@temp@fam\}

\font@rst\and famrst\ are only used when there is no global settings, to save and restore different families. Not really necessary, but done for optimization.
\def\bbl@font@rst#1#2#3#4{
\@ifnextchar[{bl@font@set{#1}#2#3}{#4}}
\long\def\bbl@font@set#1#2#3#4{\let\bbl@font@set@#2@#3@}

The default font families. They are eurocentric, but the list can be expanded easily with \texttt{\babelfont}.
\def\bbl@font@fams{rm,sf,tt}

⟨⟨ Font selection ⟩⟩

9 Hooks for XeTeX and LuaTeX

9.1 XeTeX

Unfortunately, the current encoding cannot be retrieved and therefore it is reset always to utf8, which seems a sensible default.
Now, the code.

\selectlanguage@x{\bbl@main@language}%
\bbl@fn@footnote\#2\#1{\ignorespaces\#3}%
\egroup}
\long\def\bbl@footnote@o#1#2#3[#4]{%
  \bgroup
  \selectlanguage@x{\bbl@main@language}%
  \bbl@fn@footnote[#4]{#2#1{\ignorespaces#3}#3}%
  \egroup}
\def\bbl@footnotetext#1#2#3{%
  \@ifnextchar[{
    \bbl@footnotetext@o{#1}{#2}{#3}}%
  {\bbl@footnotetext@x{#1}{#2}{#3}}}
\long\def\bbl@footnotetext@x#1#2#3#4{%
  \bgroup
  \selectlanguage@x{\bbl@main@language}%
  \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
  \egroup}
\long\def\bbl@footnotetext@o#1#2#3[#4]{%
  \bgroup
  \selectlanguage@x{\bbl@main@language}%
  \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#3}#3}%
  \egroup}
\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{#3}{#4}}%
    \@namedef{\bbl@stripslash#1text}{\bbl@footnotetext{#3}{#4}}%
  }{%
    \def#1{\bbl@exp{\bl@footnote{#3}{#4}}{#4}}%
    \@namedef{\bbl@stripslash#1text}{\bbl@exp{\bl@footnotetext{#3}{#4}}{#4}}%
  }%
\ifin@
  \bbl@ifunset{bbl@intsp@\languagename}{% 
  \expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else 
    \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\@@} \fi 
  \ifx\bbl@KVP@intrapenalty\@nnil 
    \bbl@intrapenalty0\@@ 
  \fi 
  \ifx\bbl@KVP@intraspace\@nnil \else % We may override the ini 
    \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@ 
  \fi 
  \ifx\bbl@KVP@intrapenalty\@nnil \else 
    \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@ 
  \fi 
  \bbl@exp{\% TODO. Execute only once (but redundant): 
    \bbl@add\<extras\languagename>{\XeTeXlinebreaklocale "\bbl@cl{tbcp}" \<\bbl@xeisp@\languagename> \<\bbl@xeipn@\languagename>} 
    \bbl@toglobal\<extras\languagename> 
    \bbl@add\<noextras\languagename>{\XeTeXlinebreaklocale ""} 
    \bbl@toglobal\<noextras\languagename>} \fi} \fi
\fi
\ifx\DisableBabelHook\@undefined\endinput\fi
\AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
\AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
\DisableBabelHook{babel-fontspec}
⟨⟨ Font selection ⟩⟩
\def\bbl@provide@extra#1{}

10 Support for interchar

\ifnum\xe@alloc@intercharclass<\thr@@ 
  \xe@alloc@intercharclass\thr@@ 
\fi 
\chardef\bbl@xeclass@default@=\z@ 
\chardef\bbl@xeclass@cjkideogram@=\@ne 
\chardef\bbl@xeclass@cjkleftpunctuation@=\tw@ 
\chardef\bbl@xeclass@cjkrightpunctuation@=\thr@@ 
\chardef\bbl@xeclass@boundary@=4095 
\chardef\bbl@xeclass@ignore@=4096 

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

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

\newcommand{\bbl@cmd@IfBabelInterchar}{[1]{%}
\let\bbl@cmd@tempa\@gobble % Assume to ignore
\edef\bbl@cmd@tempb{\zap@space#1 \@empty}%
\ifx\bbl@KVP@interchar\@nnil\else
\bbl@replace\bbl@KVP@interchar{ }{,}%
\bbl@foreach\bbl@cmd@tempb{%
\bbl@xin{,##1,}{,\bbl@KVP@interchar,}%
\ifin%
\let\bbl@cmd@tempa\@firstofone%
\fi%
\fi%
\fi%}
\bbl@tempa%
\newcommand{\bbl@cmd@bbl@charclass}{[3]{%}
\EnableBabelHook{babel-interchar}%
\bbl@csarg\newXeTeXintercharclass{xeiclass@#2@#1}%
\def\bbl@tempb{#1}{%}
\ifx#1\empty\else
\ifx#1-%
\bbl@upto%
\else
\bbl@charclass{%
\ifcat\noexpand#1\relax\bbl@stripslash#1\relax\string#1\fi%
\fi%
\expandafter\bbl@tempb%
\fi%}
\bbl@ifunset{bbl@xechars@#1}{%
\toks0{%}
\bbl@savevariable{XeTeXinterchartokenstate}%
\XeTeXinterchartokenstate@\@ne%
}}%
\toks0{\expandafter\expandafter\expandafter{\csname bbl@xechars@#1\endcsname}}%
\bbl@csarg\edef{xechars@#1}{%}
\toks0{%
\bbl@savevariable{XeTeXinterchartokenstate}%
\XeTeXinterchartokenstate@\@ne%
}}%
\bbl@csarg\edef{xechars@#1}{%}
\toks0{%
\bbl@savevariable{XeTeXinterchartokenstate}%
\XeTeXinterchartokenstate@\@ne%
}}%
\bbl@csarg\edef{xechars@#1}{%}
\toks0{%
\bbl@savevariable{XeTeXinterchartokenstate}%
\XeTeXinterchartokenstate@\@ne%
}}%
\bbl@csarg\edef{xechars@#1}{%}
\toks0{%
\bbl@savevariable{XeTeXinterchartokenstate}%
\XeTeXinterchartokenstate@\@ne%
}}%
\bbl@csarg\edef{xechars@#1}{%}
\toks0{%
\bbl@savevariable{XeTeXinterchartokenstate}%
\XeTeXinterchartokenstate@\@ne%
}}%
\bbl@csarg\edef{xechars@#1}{%}
\toks0{%
\bbl@savevariable{XeTeXinterchartokenstate}%
\XeTeXinterchartokenstate@\@ne%
}}%
\bbl@csarg\edef{xechars@#1}{%}
\toks0{%
\bbl@savevariable{XeTeXinterchartokenstate}%
\XeTeXinterchartokenstate@\@ne%
}}%
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>.

10.1 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 mechanism the following constructs are valid: \adim{\bbl@startskip}, \advance{\bbl@startskip}{\adim{...}}.

Consider \providetexbabel as a shorthand for \texttt{tex\-xet}, 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).

\begin{verbatim}
\def\bbl@provide@extra#1{%
% == auto-select encoding ==
\ifx\bbl@encoding@select@off\@empty\else
\bbl@ifunset{bbl@encoding@#1}%
{\def\@elt##1{,##1,}%
\edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
\count@\z@
\bbl@foreach\bbl@tempe{%
\def\bbl@tempd{##1}% Save last declared
\advance\count@\@ne}%
\ifnum\count@>\@ne % (1)
\getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
\ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
\bbl@replace\bbl@tempa{ }{,}%
\global\let\bbl@csarg\let{encoding@#1}\@empty
\bbl@xin@{,\bbl@tempd,}{,\bbl@tempa,}%
\ifin@\else % if main encoding included in ini, do nothing
\let\bbl@tempb\relax
\bbl@foreach\bbl@tempa{%
\ifx\bbl@tempb\relax
\bbl@xin@{,##1,}{,\bbl@tempe,}%
\ifin@\def\bbl@tempb{##1}\fi
\fi}
\ifx\bbl@tempb\relax\else
\bbl@exp{%
\global\let\bbl@add\let{encoding@#1}{}
\bbl@encoding@#1{\f@encoding\bbl@add}{\originalTeX{\selectfont}\fontencoding{\bbl@tempb}}\selectfont}}%
\fi%
\fi
\fi\bbl@tempb\relaxelse
\bbl@exp{%\global\let\bbl@add\let{encoding@#1}{}%}
\bbl@encoding@#1{\f@encoding{encoding@#1}}%\global\let\bbl@add\let{encoding@#1}{}
\bbl@encoding@#1{\f@encoding\bbl@add}{\originalTeX{\selectfont}\fontencoding{\bbl@tempb}}\selectfont
\bbl@tempb\relaxelse
\bbl@exp{%
\global\let\bbl@add\let{encoding@#1}{}
\bbl@encoding@#1{\f@encoding\bbl@add}{\originalTeX{\selectfont}\fontencoding{\bbl@tempb}}\selectfont
\bbl@tempb\relaxelse
\bbl@exp{%
\global\let\bbl@add\let{encoding@#1}{}
\bbl@encoding@#1{\f@encoding\bbl@add}{\originalTeX{\selectfont}\fontencoding{\bbl@tempb}}\selectfont
\fi
\fi
\fi%
\fi%
\fi}
\fi}
\end{verbatim}

10.3 LuaTeX

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

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

The default setup preloads the first language into the format. This is intended mainly for 'english', so that it's available without further intervention from the user. To avoid duplicating it, the following rule applies: if the “0th” language and the first language in language.dat have the same name then just ignore the latter. If there are new synonymous, 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 luatex patterns are best loaded when the document is typeset, and the “0th” language is preloaded just for backwards compatibility.

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

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

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

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

\begin{verbatim}
\ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
\bbl@trace{Read language.dat}
\ifx\bbl@readstream\@undefined
\csname newread\endcsname\bbl@readstream
\fi
\begingroup
\toks@{}\count@\z@ % 0=start, 1=0th, 2=normal
\def\bbl@process@line#1#2 #3 #4 {%
  \ifx=#1%
  \bbl@process@synonym{#2}%
  \else
  \bbl@process@language{#1#2}{#3}{#4}%
  \fi
\ignorespaces}
\def\bbl@manylang{
  \ifnum\bbl@last>\@ne
  \bbl@info{Non-standard hyphenation setup}%
  \fi
\let\bbl@elt\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
\chardef\bbl@last\allocationnumber
\bbl@anylang
\let\bbl@elt\relax
\fi
\endgroup
\toks@{}% 0=start, 1=0th, 2=normal
\def\bbl@process@line#1#2 #3 #4 {%
\ifx=#1%
\bbl@process@synonym{#2}%
\else
\bbl@process@language{#1#2}{#3}{#4}%
\fi
\ignorespaces}
\def\bbl@manylang{
\ifnum\bbl@last>\@ne
\bbl@info{Non-standard hyphenation setup}%
\fi
\let\bbl@elt\relax}
\def\bbl@process@language#1#2#3{%
\ifcase\count@
\or
\count@\tw@
\fi
\ifnum\count@=\tw@
\expandafter\addlanguage\csname l@#1\endcsname
\language\allocationnumber
\chardef\bbl@last\allocationnumber
\bbl@anylang
\let\bbl@elt\relax
\fi
\endgroup
\end{verbatim}

110
\begin{verbatim}
\expandafter\xdef\csname bbl@hyphendata@\the\language\endcsname
\expandafter\expandafter\expandafter\bbl@tempb
\csname bbl@hyphendata@\the\language\endcsname}}
\endinput\fi
% Here stops reading code for hyphen.cfg
% The following is read the 2nd time it's loaded
\begingroup % TODO - to a lua file
\catcode`\%=12
\catcode`\'=12
\catcode`\:=12
\directlua{
Babel = Babel or{}
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.bytes)
  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(['\[
        \csname \csname \space \bbl@info\endcsname\endcsname
        \textbf{New pattern: }\]
        .. p .. ''])
      end
      pats = pats .. ' ' .. p
    else
      tex.sprint(['\[
        \csname \csname \space \bbl@info\endcsname\endcsname
        \textbf{Renew pattern: }\]
        .. p .. ''])
      end
  end
  lang.patterns(lg, pats)
end
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 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(['\[
        \csname \csname \space \bbl@info\endcsname\endcsname
        \textbf{New pattern: }\]
        .. p .. ''])
    end
    break
  end
  tex.sprint(['\[
    \csname \csname \space \bbl@info\endcsname\endcsname
    \textbf{Renew pattern: }\]
    .. p .. ''])
end
lang.patterns(lg, pats)
end
Babel.characters = Babel.characters or{}
Babel.ranges = Babel.ranges or{}
function Babel.hlist_has_bidi(head)
  local has_bidi = false
  local ranges = Babel.ranges
end
\end{verbatim}
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
    end
    if dir and (dir == 'al' or dir == 'r') then
      has_bidi = true
    end
  end
end
return has_bidi
end
function Babel.set_chranges (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(\indexentry) and
  str:find(\babelsublr) then
    str = str:gsub(\babelsublr%s*(%b%),
    function(m) return m:sub(2,-2) end )
  end
  return str
end

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.
10.4 Southeast Asian scripts

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

\begin{verbatim}
\directlua{
  Babel = Babel or {}
  Babel.linebreaking = Babel.linebreaking or {}
  Babel.linebreaking.before = {}
  Babel.linebreaking.after = {}
  Babel.locale = {} % Free to use, indexed by \localeid
  function Babel.linebreaking.add_before(func, pos)
    tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
    if pos == nil then
      table.insert(Babel.linebreaking.before, func)
    else
      table.insert(Babel.linebreaking.before, pos, func)
    end
  end
  function Babel.linebreaking.add_after(func)
    tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
    table.insert(Babel.linebreaking.after, func)
  end
}\end{verbatim}

\begin{verbatim}
\def\bbl@intraspace#1 #2 #3\@@{%} \
\directlua{
  Babel = Babel or {}
  Babel.intraspaces = Babel.intraspaces or {}
  Babel.intraspaces[\csname bbl@sbcp@\languagename\endcsname] = %
  \{b = #1, p = #2, m = #3\}
  Babel.locale_props[\the\localeid].intraspace = %
  \{b = #1, p = #2, m = #3\}
}\end{verbatim}

\begin{verbatim}
\def\bbl@intrapenalty#1\@@{%} \
\directlua{
  Babel = Babel or {}
  Babel.intrapenalties = Babel.intrapenalties or {}
  Babel.intrapenalties[\csname bbl@sbcp@\languagename\endcsname] = #1
  Babel.locale_props[\the\localeid].intrapenalty = #1
}\end{verbatim}

\begin{verbatim}
\begin{group}
\catcode`%=12
\catcode`^=14
\catcode`\*=12
\gdef\bbl@seaintraspace{^}
\let\bbl@seaintraspace\relax
\directlua{
  Babel = Babel or {}
  Babel.sea_enabled = true
  Babel.sea_ranges = Babel.sea_ranges or {}
  function Babel.set_chranges (script, chrng)
    local c = 0
    for s, e in string.gmatch(chrng..' ', '(.-)%.(.-)%s') do
      Babel.sea_ranges[script..c] = {tonumber(s,16), tonumber(e,16)}
      c = c + 1
    end
  end
  function Babel.sea_disc_to_space (head)
    local sea_ranges = Babel.sea_ranges
    local last_char = nil
\end{verbatim}

116
local quad = 655360 % 10 pt = 655360 = 10 * 65536
for item in node.traverse(head) do
    local i = item.id
    if i == node.id'glyph' then
        last_char = item
    elseif i == 7 and item.subtype == 3 and last_char
        and last_char.char > 0x0C99 then
        quad = font.getfont(last_char.font).size
        for lg, rg in pairs(sea_ranges) do
            if last_char.char > rg[1] and last_char.char < rg[2] then
                lg = lg:sub(1, 4) % Remove trailing number of, eg, Cyril
                local intraspace = Babel.intraspaces[lg]
                local intrapenalty = Babel.intrapenalties[lg]
                local n
                if intrapenalty ~= 0 then
                    n = node.new(14, 0) % penalty
                    n.penalty = intrapenalty
                    node.insert_before(head, item, n)
                end
                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
end
end
}
\bbl@luahyphenate

10.5 CJK line breaking

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

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

\catcode`%=14
\edef\bbl@cjkintraspace{%
\let\bbl@cjkintraspace\relax
\directlua{
Babel = Babel or {
require('babel-data-cjk.lua')
Babel.cjk_enabled = true
function Babel.cjk_linebreak(head)
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]
end
}^^
\bbl@luahyphenate}
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 % CL
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 intrapenalty = props.intrapenalty
  if intrapenalty == 0 then
    local n = node.new(14, 0) % penalty
    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
else % if penalty, glue or anything else
  last_class = nil
end

lang.hyphenate(head)

}%
\bbl@luahyphenate}
\gdef\bbl@luahyphenate{%
\let\bbl@luahyphenate\relax
\directlua{
  luatexbase.add_to_callback('hyphenate',
    function (head, tail)
      if Babel.linebreaking.before then
        for k, func in ipairs(Babel.linebreaking.before) do
          func(head)
        end
      end
      if Babel.cjk_enabled then
        Babel.cjk_linebreak(head)
      end
      lang.hyphenate(head)
      if Babel.linebreaking.after then
        for k, func in ipairs(Babel.linebreaking.after) do
          func(head)
        end
      end

end}

118
end
if Babel.\texttt{sea\_enabled} then
  Babel.\texttt{sea\_disc\_to\_space}(head)
end,
'Babel.\texttt{hyphenate}')
}
\endgroup
\def\bbl@provide@intraspace{%\if\bbl@isunset{bbl@intsp\@\languagename}{}%\
\ifx\texttt{bbl@intsp\@\languagename}\@empty\else\
\bbl@xin{/c}{/\bbl@cl{lnbrk}}%\ifin@ % cjk\
\bbl@cjkintraspace\directlua{\texttt{Babel = Babel or {}\n  Babel.locale\_props = Babel.locale\_props or {}\n  Babel.locale\_props[\the\localeid].linebreak = 'c'\n}}%
\bbl@exp{\texttt{\bbl@intraspace}\bbl@cl{intsp}\
\@@}\ifx\texttt{\bbl@KVP@intrapenalty}\@nnil\
\bbl@intrapenalty\texttt{0}\@@\fi%\else % sea\
\bbl@seaintraspace\directlua{\texttt{Babel = Babel or {}\n  Babel.sea\_ranges = Babel.sea\_ranges or {}\n  Babel.set\_chranges('{\texttt{bbl@cl{sbcp}}}',\n'\texttt{bbl@cl{chrng}}')}\n}}%
\ifx\texttt{\bbl@KVP@intrapenalty}\@nnil\
\bbl@intrapenalty\texttt{0}\@@\fi%\else % bidimode>100 \ifnum\bbl@bidimode<200\
\def\bblar@chars{\texttt{0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,0640,0641,0642,0643,0644,0645,0646,0647,0649}\
\def\bblar@elongated{\texttt{0626,0628,062A,062B,0633,0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,0641,0642,0643,0644,0645,0646,0647,0649}}\
\begingroup\
\catcode`_=11 \catcode`:=11\
\gdef\bblar@nofswarn{\gdef\msg\texttt{warning:nxn##1##2##3}{}}\
\endgroup\
\gdef\bbl@arabicjust{% TODO. Allow for several locales.\
\let\bbl@arabicjust\relax\
\newattribute\bblar@kashida\directlua{\texttt{Babel.attr\_kashida = \texttt{luatexbase.registernumber'bblar\@kashida'}}}\%\begin{group}\input{\texttt{\bblar@chars}}\input{\texttt{\bblar@elongated}}\end{group}\
10.6 Arabic justification
\textit{WIP.} \texttt{\bbl@arabicjust} is executed with both elongated an kashida. This must be fine tuned. The attribute \texttt{kashida} is set by transforms with kashida-
\def\bblar@fetchjalt#1#2#3#4{%
  \bbl@exp{\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
}%
}

Elongated forms. Bruteforce. Norules at all, yet. The ideal: look at jalt table. And perhaps other tables (falt?, cswh?)? What about kaf? And diacritic positioning?
\gdef\bblar@parsejalt{%
  \ifx\addfontfeature\@undefined\else
    \bbl@xin@{/e}{/\bbl@cl{lnbrk}}\ifin@
      \directlua{%if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
        Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
      end
      tex.print({[[\string\csname\space bbl@parsejalti\endcsname]]})
    }%
  \fi
  \fi
}\gdef\bblar@parsejalti{%
  \begingroup
    \let\bblar@parsejalt\relax % To avoid infinite loop
  \edef\bbl@tempb{\fontid\font}
  \bblar@nofswarn
  \bblar@fetchjalt\bblar@elongated{}{from}{}\bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
  \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
  \addfontfeature{RawFeature=+jalt}%
  \% @namedef{bblar@JE@0643}{066A}% todo: catch medial kaf
  \bblar@fetchjalt\bblar@elongated{}{dest}{}\bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
  \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
  \directlua{%for k, v in pairs(Babel.arabic.from) do
    if Babel.arabic.dest[k] and
      not {Babel.arabic.from[k] == Babel.arabic.dest[k]} then
      Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
        [Babel.arabic.from[k]] = Babel.arabic.dest[k]
    end
  end
}}
The actual justification (inspired by CHICKENIZE).

\begin{group}
\catcode`#=11
\catcode`~=11
\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(12, 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'glyph'
  local KASHIDA = Babel.attr_kashida
  local LOCALE = Babel.attr_locale

  if line == nil then
    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!
  % ? Look for glue = 12:15
  if (line.glue_sign == 1 and line.glue_order == 0) then
    elongs = {} % Stores elongated candidates of each line
    k_list = {} % And all letters with kashida
    pos_inline = 0 % Not yet used
end

121
for n in node.traverse_id(GLYPH, line.head) do
  pos_inline = pos_inline + 1 % To find where it is. Not used.
  % Elongated glyphs
  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

% Tatwil
if Babel.kashida_wts then
  local k_wt = node.get_attribute(n, KASHIDA)
  if k_wt > 0 then % todo. parameter for multi inserts
    table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
  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 curr = elongs[x].node
  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
  % Substitute back if the line would be too wide and break:
  if width > goal then
    curr.char = oldchar
    break
  end
  % If continue, pop the just substituted node from the list:
  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
  end
  wt_pos = wt_pos + 1
  k_curr = k_curr - 1
end
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
if Babel.fix_diacr then
    Babel.fix_diacr(k_item.next)
end
width = width_new
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% 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
        else
            cnt = 0
        end
    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% if process line
end} % ends Arabic just block: \ifnum\bbl@bidimode>100...

10.7 Common stuff
\AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
\AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
\DisableBabelHook{babel-fontspec}
⟨⟨Font selection⟩⟩
10.8 Automatic fonts and ids switching
After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a the function Babel.locale_map, which just traverse the node list to carry out the replacements. The table loc_to_scr stores the script range for each locale (whose id is the key),
copied from this table (so that it can be modified on a locale basis); there is an intermediate table named chr_to_loc built on the fly for optimization, which maps a char to the locale. This locale is then used to get the language as stored in locale_props, as well as the font (as requested). In the latter table a key starting with / maps the font from the global one (the key) to the local one (the value). Maths are skipped and 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'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
['Cyril'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F}, {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF}, {0xAB00, 0xAB2F}},
['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF}, {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF}, {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF}, {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F}, {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF}, {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
['Hebr'] = {{0x0590, 0x05FF}},
['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF}, {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
['Knda'] = {{0x0C80, 0x0CFF}},
['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x30FF}, {0x3130, 0x318F}, {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF}, {0xDF00, 0xE1FF}},
['Laoo'] = {{0x0E80, 0x0EFF}},
['Latn'] = {{0x0000, 0x00FF}, {0x0100, 0x017F}, {0x0180, 0x01FF}, {0x2C60, 0x2C7F}, {0xA720, 0xA77F}, {0x0AB0, 0x0AB6F}},
['Mahj'] = {{0x01150, 0x0117F}},
['Mlym'] = {{0x0D00, 0x0D7F}},
['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA960, 0xA97F}},
['Orya'] = {{0x0B00, 0x0B7F}},
['Syrc'] = {{0x0700, 0x074F}, {0x0800, 0x086F}},
['Tam'] = {{0x0B80, 0x0BF}},
['Telu'] = {{0x0C00, 0x0C7F}},
['Tfng'] = {{0x2D30, 0x2D7F}},
['Thai'] = {{0xE900, 0xE7F}},
['Tibt'] = {{0x0F00, 0xFFF}},
['Vai'] = {{0xA500, 0xA63F}},
['Yi'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
}
Babel.script_blocks.Cyrs = Babel.script_blocks.Cyril
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
    end
  end
  % Treat composite chars in a different fashion, because they
  % 'inherit' the previous locale.
  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
  if toloc == -2000 then
    toloc = toloc_save
  elseif toloc == -1000 then
    toloc = nil
  end
  if toloc and Babel.locale_props[toloc] and
    Babel.locale_props[toloc].letters and
    tex.getcatcode(item.char) ~= 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
  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)
  end
end
6144   elseif item.id == node.id.match then
6145       inmatch = (item.subtype == 0)
6146   end
6147 end
6148 return head
6149 end
6150}

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

6151 newcommand\babelcharproperty[1]{%
6152 \count@=#1\relax
6153 \ifvmode
6154 \expandafter\bbl@chprop
6155 \else
6156 \bbl@error[charproperty-only-vertical]{}
6157 \fi
6158 newcommand\bbl@chprop[3][\the\count@]{%
6159 \@tempcnta=#1\relax
6160 \bbl@ifunset{\bbl@chprop@#2}{}{unknown-char-property}
6161 \bbl@error[unknown-char-property]\{}\{#2\}\%
6162 \}
6163 \loop
6164 \bbl@cs{chprop@#2}\%
6165 \ifnum\count@<\@tempcnta
6166 \advance\count@\@ne
6167 \repeat}
6168 \def\bbl@chprop@direction#1{%
6169 \directlua{
6170 Babel.characters[\the\count@] = Babel.characters[\the\count@] or {};
6171 Babel.characters[\the\count@][\textquotesingle d\textquotesingle] = \textquotesingle #1\textquotesingle
6172 }
6173 \let\bbl@chprop@bc\bbl@chprop@direction
6174 \def\bbl@chprop@mirror#1{%
6175 \directlua{
6176 Babel.characters[\the\count@] = Babel.characters[\the\count@] or {};
6177 Babel.characters[\the\count@][\textquotesingle m\textquotesingle] = \textquotesingle number\textquotesingle
6178 }
6179 \let\bbl@chprop@bmg\bbl@chprop@mirror
6180 \def\bbl@chprop@linebreak#1{%
6181 \directlua{
6182 Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {};
6183 Babel.cjk_characters[\the\count@][\textquotesingle c\textquotesingle] = \textquotesingle #1\textquotesingle
6184 }
6185 \let\bbl@chprop@lb\bbl@chprop@linebreak
6186 \def\bbl@chprop@locale#1{%
6187 \directlua{
6188 Babel.char_to_loc = Babel.char_to_loc or {};
6189 Babel.char_to_loc[\the\count@] =
6190 \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id}@id}1\space
6191 }
6192 \let\bbl@chprop@locale\bbl@chprop@locale
6193 \def\bbl@chprop@locale\bbl@chprop@locale
6194 }

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.

6195 \def\bbl@chprop@locale\bbl@chprop@locale
6196 \def\bbl@chprop@locale\bbl@chprop@locale
6197 }

Now the \LaTeX high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the \{n\} syntax. For example, \texttt{pre=\{1\}\{1\}\{-1000\}} becomes \texttt{function(m) return m[1]..m[1]..'-' end}, where \texttt{m} 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 load – save the code as string in a TeX macro, and expand this macro at the appropriate place. As \directlua does not take into account the current catcode of \&, we just avoid this character in macro names (which explains the internal group, too).

\begin{verbatim}
6195 \begingroup
6196 \catcode`\%=12
6197 \catcode`\&=14
6198 \catcode`\|=12
6199 \gdef\babelprehyphenation{&% 
6200 \@ifnextchar[{bl@settransform{0}}{bl@settransform{0}[]}}
6201 \gdef\babelposthyphenation{&% 
6202 \@ifnextchar[{bl@settransform{1}}{bl@settransform{1}[]}}
6203 \gdef\bbl@settransform[#1][#2]\#3\#4\#5{&% 
6204 \ifcase#1
6205 \bbl@activateprehyphen
6206 \or
6207 \bbl@activateposthyphen
6208 \fi
6209 \fi
6210 \begingroup
6211 \def\bbl@add@list\babeltempb{&% 
6212 \let\babeltempb\@empty
6213 \def\bbl@tempa{#5}&% 
6214 \bbl@replace\bbl@tempa{,}{ ,}&% TODO. Ugly trick to preserve {}
6215 \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&% 
6216 \bbl@ifsamestring{##1}{remove}{&% 
6217 \bbl@add@list\babeltempb{nil}}&% 
6218 \directlua{
6219 local rep = 
6220 rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6221 rep = rep:gsub('^%s*(insert)%s*$', 'insert = true ,')
6222 rep = rep:gsub(('string')%s*=%s*(["%s","])', babel.capture_func)
6223 if #1 == 0 or #1 == 2 then
6224 rep = rep:gsub(('space')%s*=%s*(["%d."]+)\%s+(["%d."]+)\%s+(["%d."]+)','
6225 space = [ .. \%2, \%3, \%4 ... ]')
6226 rep = rep:gsub(('spacefactor')%s*=%s*(["%d."]+)\%s+(["%d."]+)\%s+(["%d."]+)','
6227 spacefactor = [ .. \%2, \%3, \%4 ... ]')
6228 rep = rep:gsub(('kashida')%s*=%s*(["%s","])', babel.capture_kashida)
6229 else
6230 rep = rep:gsub( '(no)%s*=%s*(["%s","])', babel.capture_func)
6231 rep = rep:gsub( '(pre)%s*=%s*(["%s","])', babel.capture_func)
6232 rep = rep:gsub( '(post)%s*=%s*(["%s","])', babel.capture_func)
6233 end
6234 tex.print([[\string\babeltempa{{["}}} .. rep .. [[}}}])&% 
6235 }&%
6236 \bbl@foreach\babeltempb&% 
6237 \bbl@forkv{[#1]}&% 
6238 \in{,###1},[nil,step,remove,string,no,pre,\%
6239 no,pre,penalty,kashida,space,spacefactor,}&% 
6240 \ifin
6241 \bbl@error{bad-transform-option}{###1}{{}&% 
6242 \fi}&% 
6243 \let\bbl@kv@attribute\relax
6244 \let\bbl@kv@label\relax
6245 \let\bbl@kv@fonts\empty
6246 \bbl@forkv[#2]\bbl@csarg\edef{kv@###1}[#2]}&% 
6247 \ifx\bbl@kv@fonts\empty\else\bbl@settransform{fi
6248 \ifx\bbl@kv@attribute\relax
6249 \ifx\bbl@kv@label\relax\else
6250 \bbl@exp{\\bbl@trim\def\\bbl@kv@fonts{\bbl@kv@fonts}}&% 
6251 \bbl@replace\bbl@kv@fonts{()},&% 
6252 \edef\bbl@kv@attribute{\bbl@ATR@\\bbl@kv@label @#3@\\bbl@kv@fonts}&% 
6253 \count\v@&% 
6254 \def\bbl@elt#1#2#3{&% 

\end{verbatim}
\end{verbatim}

127
\ifsamestring{#3}{#1}{##1,##2}&\%
\ifsamestring{#3}{#2}&\%
\count@\@ne&\%
\error{font-conflict-transforms}{}{}{}}&\%
\transfont@list
\exp{\global\\add\\transfont@list}{\\elt{#3}{#1}{#2}}&\%
\ifunset{\kv@attribute}&\%
\global\cpp\setattribute{#1}{}&\%
\else&\%
\edef\kv@attribute{\expandafter\stripslash\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
\if\bbl@kv@attribute\relax
attr = -1
\else
attr = luatexbase.registernumber'\bbl@kv@attribute'
\fi
\if\bbl@kv@label\relax&\%
label = [[\bbl@kv@label]\]]\]
\fi
\relax&\%
\global\cpp\atcarg\newattribute{\bbl@kv@attribute}\@ne
\else&\%
\edef\kv@attribute{\expandafter\stripslash\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
\if\bbl@kv@attribute\relax
attr = -1
\else
attr = luatexbase.registernumber'\bbl@kv@attribute'
\fi
\if\bbl@kv@label\relax&\%
label = [[\bbl@kv@label]\]]\]
\fi
\relax&\%
\global\cpp\atcarg\newattribute{\bbl@kv@attribute}\@ne
}
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 \[ \text{[==]} \text{\]}). 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.

10.9 Bidi

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

\newcommand{\localereprehyphenation}[1]{%  
\directlua{  
\localereprehyphenation[1]}%}

10.9 Bidi

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

\newcommand{\localereprehyphenation}[1]{%  
\directlua{  
\localereprehyphenation[1]}%}
\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
end
%
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
end
%
luatexbase.add_to_callback('pre_linebreak_filter',
    Babel.pre_otfload_v,
    'Babel.pre_otfload_v',
    luatexbase.priority_in_callback('pre_linebreak_filter',
        'luaotfload.node_processor') or nil)
%
luatexbase.add_to_callback('hpack_filter',
    Babel.pre_otfload_h,
    'Babel.pre_otfload_h',
    luatexbase.priority_in_callback('hpack_filter',
        'luaotfload.node_processor') or nil)
}

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=.

\breakafterdirmode=1
\ifnum\bbl@bidimode=\@ne % 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
\chardef\bbl@getluadir#1{\@}
\chardef\bbl@pdir#1@
\def\bbl@getluadir#1{%
\directlua{
    if tex.#1dir == 'TLT' then
        tex.sprint('0')
    elseif tex.#1dir == 'TRT'

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 bidi=basic, without having to patch almost any macro where text direction is relevant.
Still, there are three areas deserving special attention, namely, tabular, math, and graphics, text and intrinsically left-to-right elements are intermingled. I've made some progress in graphics, but they're essentially hacks; I've also made some progress in 'tabular', but when I decided to tackle math (both standard math and 'amsmath') the nightmare began. I'm still not sure how 'amsmath' should be modified, but the main problem is that, boxes are "generic" containers that can hold text, math, and graphics (even at the same time; remember that inline math is included in the list of text nodes marked with 'math' (11) nodes too).

@hangfrom is useful in many contexts and it is redefined always with the layout option. There are, however, a number of issues when the text direction is not the same as the box direction (as set by \bodydir), and when \parbox and \hangindent are involved. Fortunately, latest releases of luatex simplify a lot the solution with \shapemode.

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

6481 \bbl@trace{Redefinitions for bidi layout}
6482 %
6483 ⟨⟨More package options⟩⟩
6484 \chardef\bbl@eqnpos\z@
6485 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6486 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6487 ⟨⟨More package options⟩⟩
6488 %
6489 \ifnum\bbl@bidimode\z@ % Any bidi=
6501 \let\bbl@mathboxdir\relax
6502 \def\bbl@eqdel{()}
6503 \def\bbl@eqno@flip#1{%
6504 \ifdim\predisplaysize=-\maxdimen
6505 \leqno
6506 \hb@xt@.01pt{%04\@ \\
6507 \hss\hb@xt@0.01pt{
6508 \leqno\hbox{\displaywidth{\hss{\glet\bbl@upset\@currentlabel}}}}%04\@ \\
6509 \if\fi
6510 \def\bbl@eqnum{\leqno{\hbox{\theequation}}}
6511 \if\fi
6512 \hb@xt@.01pt{%04\@ \\
6513 \leqno\hbox{\displaywidth{\hss{\glet\bbl@upset\@currentlabel}}}}%04\@ \\
6514 \if\fi
6515 \def\bbl@eqdel{()}
6516 \AtBeginDocument{%04\@ \\
6517 \if\fi
6518 \if\fi
6519 \bbl@add\normalfont{\bbl@eqnodir}
inmath = (item.subtype == 0)
end
end
return head
end
}

% == transforms ==
def\bbl@elt##1##2##3{\in@{\transforms.}{##1}}%
def\bbl@tempa{\transforms.}{}%
\bbl@carg{\bbl@transforms}{ babel\bbl@tempa}{##2}{##3}%
\csname bbl@inidata@\languagename\endcsname
\bbl@release@transforms\relax % \relax closes the last item.

% Start tabular here:
def\localerestoredirs{%
\ifcase\bbl@thetextdir
  \ifnum\textdirection=\z@\else\textdir TLT\fi
\else
  \ifnum\textdirection=\@ne\else\textdir TRT\fi
\fi
\ifcase\bbl@thepardir
  \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
\else
  \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
\fi
\IfBabelLayout{tabular}%
  % All RTL
  % Mixed, with LTR cols
  \let\bbl@parabefore\relax
  \AtBeginDocument{%
    \bbl@replace@tabular[]% Hide conditionals
    \bbl@exp{\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs%}
    \bbl@exp{\bgroup\bgroup\bgroup\localerestoredirs%}
    % Any lua bidi= except default=1
    \bbl@replace@tabular[]% Hide conditionals
    \bbl@exp{\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs%}
    \bbl@exp{\bgroup\bgroup\bgroup\localerestoredirs%}
    \bbl@exp{\bgroup\bgroup\bgroup\localerestoredirs%}
    % \do@row@strut%}
  }%
  % \do@row@strut
  %}%
  %}%
}
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.

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

\ifnum\bbl@bidimode<0 % Any bidi=
  \def\bbl@nextfake#1{% non-local changes, use always inside a group!
    \bbl@exp{% 
      \def\\bbl@insidemath{0} \mathdir \the\bodydir #1% Once entered in math, set boxes to restore values
      \bbl@nextfake*% 
      \everyvbox{%
        \the\everyvbox \bodydir\the\bodydir \mathdir\the\mathdir \everyvbox\{\the\everyvbox\}%
      }%}
  }
\fi
\IfBabelLayout{tabular}{\let\bbl@OL@@tabular\@tabular
  \bbl@replace\@tabular{$}\bbl@nextfake$%}
\let\bbl@NL@@tabular\@tabular
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.

\IfBabelLayout{counters}{\bbl@add\bbl@opt@layout{.counters}}\def\directlua{luatexbase.add_to_callback("process_output_buffer",Babel.discard_sublr,"Babel.discard_sublr")}
\IfBabelLayout{counters*}{\bbl@add\bbl@opt@layout{.counters}}
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.

```
\IfBabelLayout{footnotes}{
  \let\bbl@OL@labelenumii\labelenumii
  \def\labelenumii{)
    \theenumii(}
\}

\IfBabelLayout{extras}{
  \let\bbl@OL@underline{underline}
  \bbl@carg\bbl@sreplace{underline}{\bgroup\bbl@nextfake$\underline$}
  \bbl@carg\bbl@sreplace{underline}{$}{\m@th$}
  \let\bbl@OL@LaTeXe\LaTeXe
  \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
    \if b\expandafter\@car\f@series\@nil\boldmath\fi
    \if b\expandafter\@car\f@series\@nil\boldmath\fi
    \bbl@ncarg\let\bbl@OL@footnote\footnote
    \BabelFootnote\footnote\languagename{}
    \BabelFootnote\localfootnote\languagename{}
    \BabelFootnote\mainfootnote{}
  }
```

### 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 discretionary patterns 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 \texttt{luatex} manual), we must convert it to a utf8 position. With \texttt{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 \texttt{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.

```
10.11 Lua: transforms

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
      last = n
    end
  end
end
```
return head

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 innmath = false

  while item do
    if item.id == 11 then
      innmath = (item.subtype == 0)
    end
    if innmath then
      -- pass
    elseif item.id == 29 then
      local locale = node.get_attribute(item, Babel.attr_locale)
      if lang == locale or lang == nil then
        lang = lang or locale
      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
    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
  end

  -- 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)
    word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
  end
  return word_string, word_nodes, item, lang
end
local word_string = ''
local word_nodes = {}
local lang
local item = head
local inmath = false
while item do
  if item.id == 11 then
    inmath = (item.subtype == 0)
  end
  if inmath then
    -- pass
  elseif item.id == 29 then
    if item.lang == lang or lang == nil then
      if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
        lang = lang or item.lang
        word_string = word_string .. unicode.utf8.char(item.char)
        word_nodes[#word_nodes+1] = item
      end
    else
      break
    end
  elseif item.id == 7 and item.subtype == 2 then
    word_string = word_string .. '='
    word_nodes[#word_nodes+1] = item
  elseif item.id == 7 and item.subtype == 3 then
    word_string = word_string .. '|
    word_nodes[#word_nodes+1] = item
  -- (1) Go to next word if nothing was found, and (2) implicitly
  -- remove leading USs.
  elseif word_string == '' then
    -- pass
  -- This is the responsible for splitting by words.
  elseif (item.id == 12 and item.subtype == 13) then
    break
  else
    word_string = word_string .. Babel.us_char
    word_nodes[#word_nodes+1] = item -- Will be ignored
  end
  item = item.next
word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
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)
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, mw, lang = Babel.fetch_subtext[mode](word_head)
        if Babel.debug then
            print()
            print((mode == 0) and '@@@@<' or '@@@@>', w)
        end
        if mw == 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)
            end
            local last_match = 0
            local step = 0
            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
                -- Get and remove empty captures (with {}'s, which return a
                -- number with the position), and keep actual captures
                -- (from (...)), if any, in matches.
                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

-- This loop stores in a small table the nodes
-- corresponding to the pattern. Used by 'data' to provide a
-- predictable behavior with 'insert' (w.nodes is modified on
-- the fly), and also access to 'remove'd nodes.
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
  sc = sc + 1
  rc = rc + 1
end

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

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

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

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
                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 ''
    last_match = utf8.offset(w, sc+1+step)
    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
    end

144
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
    else 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, spaceskip)
        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, spaceskip)
        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  

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 = mw
end -- for substring
return head
end -- for substring

-- 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("%.%", '')
  return key .. "]=function(m) return " .. ret .. " end"
end

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%x+)", function (n)
    return u.char(tonumber(n, 16))
  end)
  to = u.gsub(to, "{(%x%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 "]=function(m) return " .. capno .. "]", ..
end
Create/Extend reversed sorted list of kashida weights:

```lua
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).

```lua
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)
    else
      n = node.new(29, 0)
    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 .. node.get_attribute(n, Babel.attr_locale)
    elseif n.id == 29 then
      res = res .. unicode.utf8.char(n.char)
    end
  end
  tex.print(res)
end
```

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', m=0x29},
[0x28]={d='on', m=0x29},
```

147
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):

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

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, what they do and why, and not only how), but I think (or I hope) I’ve managed to understand them.

In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually two R modes (set explicitly in Unicode with RLM and ALM). In babel the dir is set by a higher protocol based on the language/script, which in turn sets the correct dir (<l>, <r> or <al>), From UAX#9: “Where available, markup should be used instead of the explicit formatting characters”. So, this simple version just ignores formatting characters. Actually, most of that annex is devoted to how to handle them.

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

```
local characters = Babel.characters
local ranges = Babel.ranges
local DIR = node.id("dir")

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

    nextalsodependsonscript/lang(<a>|<l>/<r>). Tobesetby babel.
	next also depends on script/lang (<a>/<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]
        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
        end
        dir = dir or 'l'
        if inmath then dir = ('TRT' == tex.mathdir) and 'r' or 'l' end
        if new_dir then
            attr_dir = 0
            for at in node.traverse(item.attr) do
                if at.number == Babel.attr_dir then
                    attr_dir = at.value & 0x3
                end
            end
            if attr_dir == 1 then
                strong = 'r'
            elseif attr_dir == 2 then
                strong = 'al'
            else
                strong = 'l'
            end
            strong_lr = (strong == 'l') and 'l' or 'r'
            outer = strong_lr
            new_dir = false
        end
        if dir == 'nsm' then dir = strong end -- W1
    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).

if new_dir then
    attr_dir = 0
    for at in node.traverse(item.attr) do
        if at.number == Babel.attr_dir then
            attr_dir = at.value & 0x3
        end
    end
    if attr_dir == 1 then
        strong = 'r'
    elseif attr_dir == 2 then
        strong = 'al'
    else
        strong = 'l'
    end
    strong_lr = (strong == 'l') and 'l' or 'r'
    outer = strong_lr
    new_dir = false
end

if dir == 'nsm' then dir = strong end    -- W1

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

if dir == 'en' then dir = 'an' end -- W2
if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
strong_lr = 'r' -- W3
end

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

elseif item.id == node.id'dir' and not inmath then
        new_dir = true
        dir = nil
elseif item.id == node.id'math' then
        inmath = (item.subtype == 0)
else
        dir = nil -- Not a char
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 whatst, 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>.

if dir == 'en' or dir == 'an' or dir == 'et' then
        if dir ~= 'et' then
                type_n = dir
        end
        first_n = first_n or item
        last_n = last_es or item
        last_es = nil
elseif dir == 'es' and last_n then -- W3+W6
        last_es = item
elseif dir == 'cs' then -- it's right - do nothing
elseif 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_n, last_n, 'r')
                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 = ''
        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.

```lua
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).

```lua
if dir == 'l' or dir == 'r' then
    last_lr = item
    strong = dir_real -- Don’t search back - best save now
    strong_lr = (strong == 'l') and 'l' or 'r'
elseif new_dir then
    last_lr = nil
end
```

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

```lua
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
end
```

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

```lua
return node.prev(head) or head
end
end
```

And here the Lua code for bidi=basic:

```lua
Babel = Babel or {}

-- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
```

Babel.fontmap = Babel.fontmap or {}
Babel.fontmap[0] = {} -- l
Babel.fontmap[1] = {} -- r
To cancel mirroring. Also OML, OMS, U?

```
Babel.symbol_fonts = Babel.symbol_fonts or {}
Babel.symbol_fonts[font.id('tenln')] = true
Babel.symbol_fonts[font.id('tenlnw')] = true
Babel.symbol_fonts[font.id('tencirc')] = true
Babel.symbol_fonts[font.id('tencircw')] = true
```

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
    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
  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 = '-TLT'
    _, new = node.insert_after(head, state.ean, d)
    if state.ean == state.eim then state.eim = new end
  end
  new_state.san, new_state.ean = nil, nil
  return head, new_state
end
```

```
local function glyph_not_symbol_font(node)
  if node.id == GLYPH then
    return not Babel.symbol_fonts[node.font]
  else
    return false
  end
end
```

```
-- 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 s 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
  -- 'al' is only taken into account in the first, current loop
  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 glyph_not_symbol_font(item) 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] end
    end
    break
  end
  end
  if item.id == DIR then
    d = nil
  elseif item.id == node.id'glue' and item.subtype == 13 then
    glue_d = nil
    glue_i = nil
  end
  elseif item.id == DIR then
    d = nil
  elseif head ~= item then new_d = true end
  elseif item.id == node.id'glue' and item.subtype == 13 then
    glue_d = d
end
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
    temp = 'on' -- W6
  end
  for e = first_et, #nodes do
    if glyph_not_symbol_font(nodes[e][1]) then
      nodes[e][2] = temp
    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

-- 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
end

for e = first_et, #nodes do
  if glyph_not_symbol_font(nodes[e][1]) then
    nodes[e][2] = temp
  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
  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
end

end
for r = first_on, q - 1 do
    nodes[r][2] = temp
    item = nodes[r][1] -- MIRRORING
    if Babel.mirroring_enabled and glyph not_symbol_font(item)
        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
----------- IMPPLICIT, 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)
    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
    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
        end
    end
end
end
state.eim = item
elseif d == 'r' and state.sim then
    head, state = insert_implicit(head, state, outer)
elseif d == 'r'
    state.sim, state.eim = nil, nil
end
end
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
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
            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:

```
[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.

\ProvidesLanguage{nil}{\langle\langle\text{date}\rangle\rangle}{\langle\langle\text{version}\rangle\rangle} Nil language

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\l@nil
\let\bbl@elt\relax
\edef\bbl@languages{% Add it to the list of languages
\bbl@languages\bbl@elt\langle\langle\text{date}\rangle\rangle{\langle\langle\text{version}\rangle\rangle}\{\langle\text{hyphenation categories}\rangle\}}}
\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

\let\captionsnil\@empty
\let\datenil\@empty

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}{DFLT}%
\bbl@elt{identification}{level}{1}%
\bbl@elt{identification}{encodings}{}%
\bbl@elt{identification}{derivate}{no}%
\@namedef{bbl@tbcp@nil}{und}%
\@namedef{bbl@lbcp@nil}{und}%
\@namedef{bbl@casing@nil}{und} % TODO
\@namedef{bbl@lotf@nil}{dflt}
\@namedef{bbl@elname@nil}{nil}%
\@namedef{bbl@lname@nil}{nil}%
\@namedef{bbl@esname@nil}{Latin}%
\@namedef{bbl@sname@nil}{Latin}%
\@namedef{bbl@sbcp@nil}{Latn}%
\@namedef{bbl@sotf@nil}{latn}
The macro \texttt{\texttt{ldf@finish}} takes care of looking for a configuration file, setting the main language to be switched on at \texttt{\texttt{\begin{document}}}) and resetting the category code of \texttt{@} to its original value.

\texttt{\texttt{ldf@finish(nil)}}
\texttt{\texttt{\end{document}}}  

## 13 Calendars

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

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

\begin{verbatim}
⟨⟨∗Compute Julian day⟩⟩ \equiv
\def\bbl@fpmod#1#2{(#1-#2*floor(#1/#2))}
\def\bbl@cs@gregleap#1{\bbl@fpmod(\bbl@fpmod(\bbl@fpmod(#1, 100), 400), 4)
& \& \bbl@fpmod(\bbl@fpmod(#1, 100), 100) \& \& \bbl@fpmod(#1, 400) \ne 0)}
\def\bbl@cs@jd#1#2#3{% year, month, day
floor((\bbl@fpmod(#1-1, 4) + 365 * (\bbl@fpmod(#1-1, 1) + 1)) / 4)
+ -floor((\bbl@fpmod(#1-1, 100) / 100) + 360)
+ floor(((367 * #2) - 362) / 12) +
#3)
\end{verbatim}

### 13.1 Islamic

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

\begin{verbatim}
\ExplSyntaxOn
\ExplSyntaxOff
\@namedef{bbl@ca@islamic-civil++}{\texttt{bbl@ca@islamicvl@x{+2}}}
\@namedef{bbl@ca@islamic-civil+}{\texttt{bbl@ca@islamicvl@x{+1}}}
\@namedef{bbl@ca@islamic-civil}{\texttt{bbl@ca@islamicvl@x{}}}
\@namedef{bbl@ca@islamic-civil-}{\texttt{bbl@ca@islamicvl@x{-1}}}
\@namedef{bbl@ca@islamic-civil--}{\texttt{bbl@ca@islamicvl@x{-2}}}
\def\bbl@ca@islamicvl@x#1#2-#3-#4\@@#5#6#7{%
\edef\bbl@tempa{\fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%\edef#5{\fp_eval:n{ floor((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
\edef#6{\fp_eval:n{ min(12,ceil((\bbl@tempa-29+\bbl@cs@isltojd(#5){1}{1}))/29.5+1 ) }}%
\edef#7{\fp_eval:n{ \bbl@tempa - \bbl@cs@isltojd(#5){#6}{1} + 1 }}}
\end{verbatim}

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 \texttt{\today}, and maybe some close dates, data just covers Hijri \texttt{\sim} 1435/1460 (Gregorian \texttt{\sim} 2014/2038).

\begin{verbatim}
\def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
}
\end{verbatim}
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

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
\def\bbl@remainder#1#2#3{\relax
#3=#1\relax
\divide #3 by #2\relax
\advance #3 by #1\relax}
\newif\ifbbl@divisible
\def\bbl@checkifdivisible#1#2{\relax
\countdef\tmp=0
\bbl@remainder{#1}{#2}{\tmp}\relax
\ifnum \tmp=0
\global\bbl@divisibleture
\else
\global\bbl@divisibelfalse
\fi}
\newif\ifbbl@gregleap
\def\bbl@ifgregleap#1{\relax
\bbl@checkifdivisible{#1}{4}\relax
\ifbbl@divisible
\bbl@checkifdivisible{#1}{100}\relax
\ifbbl@divisible
\bbl@checkifdivisible{#1}{400}\relax
\else
\bbl@gregleapfalse
\fi
\else
\bbl@gregleaptrue
\fi
\else
\bbl@gregleapfalse
\fi
\fi}
\def\bbl@gregleap}{
\def\bbl@gregdayspriormonths#1#2#3{\relax
#3=\ifcase #1\or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
\bbl@ifgregleap{#2}\relax
\ifnum #1 > 2
\advance #3 by 1
\fi
\else
\bbl@gregleapfalse
\fi
\fi}
\def\bbl@gregdaysprioryears#1#2{\relax
\countdef\tmpc=4
\countdef\tmpb=2
\tmpb=#1\relax
\advance \tmpb by -1
\tmpc=\tmpb
\divide \tmpc by 4
\global\bbl@cntcommon=#2\relax
\countdef\tmpd=4
\countdef\tmpf=4
\tmpd=#1\relax
\advance \tmpd by -1
\tmpf=\tmpd
\multiply \tmpf by 365
\divide \tmpf by 4
\advance \tmpf by 2
\global\bbl@cntcommon=\tmpf\relax
\def\bbl@absfromgreg#1#2#3#4{\relax
#4=#3\relax
\def\bbl@gregdaysprioryears#1#2{\relax
\countdef\tmpc=4
\countdef\tmpb=2
\tmpb=#1\relax
\advance \tmpb by -1
\tmpc=\tmpb
\multiply \tmpc by 365
\divide \tmpc by 4
\advance \tmpc by 2
\tmpc=\tmpd
\divide \tmpc by 4
\advance \tmpc by 2
\tmpc=\tmpf
\divide \tmpc by 400
\global\bbl@cntcommon=\tmpc\relax
\global\bbl@absolute#1#2#3#4\relax
}
\ifnum \tmpc < 16789
\else
  \ifnum \tmpa=1
    \advance #1 by -1
    \bbl@checkleaphebryear{#1}% at the end of leap year
  \fi
\fi
\else
  \fi
\fi
\fi
\fi
\advance #2 by 1
\fi
\bbl@remainder{#2}{7}{\tmpa}%
\ifnum \tmpa=0
  \advance #2 by 1
\else
  \ifnum \tmpa=3
    \advance #2 by 1
  \else
    \ifnum \tmpa=5
      \advance #2 by 1
    \fi
  \fi
\fi
\global\bbl@cntcommon=#2\relax}%
#2=\bbl@cntcommon}
\def\bbl@daysinhebryear#1#2{%
{\countdef\tmpe=12
 \bbl@hebrelapseddays{#1}{\tmpe}%
 \advance #1 by 1
 \bbl@hebrelapseddays{#1}{#2}%
 \advance #2 by -\tmpe
 \global\bbl@cntcommon=#2}%
\def\bbl@hebrdayspriormonths#1#2#3{%
{\countdef\tmpf= 14
 \ifcase #1\relax
 0 \or
 0 \or
 30 \or
 59 \or
 89 \or
 118 \or
 148 \or
 177 \or
 207 \or
 236 \or
 266 \or
 295 \or
 325 \or
 400
 \fi
 \bbl@checkleaphebryear{#2}%
 \if\bbl@hebrleap
 \ifnum #1 > 6
  \advance #3 by 30
 \fi
 \fi
\bbl@daysinhebryear{#2}{\tmpf}%
13.3 Persian

There is an algorithm written in TeX by Jabri, Abolhassani, Pournader and Esfahbod, created for the first versions of the Farsi TeX system (no longer available), but the original license is GPL, so its use with LPPL is problematic. The code here follows loosely that by John Walker, which is free and accurate, but sadly very complex, so the relevant data for the years 2013-2050 have been pre-calculated and stored. Actually, all we need is the first day (either March 20 or March 21).

\ExplSyntaxOn
\def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
\def\bbl@cPersian#1-#2-#3\@@#4#5#6{\edef\bbl@tempa{#1}% 20XX-03-bl@tempe = 1 farvardin:
\ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051 \bbl@afterfi\expandafter\@gobble\fi\fi
{\bbl@error{year-out-range}{2013-2050}{}{}}\bbl@xin@\bbl@tempa{\bbl@cs@firstjal@xx}\ifin@\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi\edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
\edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin\ifnum\bbl@tempc<\bbl@tempb \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo\bbl@xin@\bbl@tempa{\bbl@cs@firstjal@xx}\ifin@\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi\edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%\fi
\edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
\edef#5{\fp_eval:n{(\bbl@tempc - ((#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30))}}}
\edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}
\ExplSyntaxOff
\fi\fi
\ExplSyntaxOn
\def\bbl@ca@coptic#1-#2-#3\@@#4#5#6{\edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}% current
\edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}% begin\edef#4{\fp_eval:n{floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}% 166
13.5 Buddhist

That's very simple.

\begin{macrocode}
\edef\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{\edef#4{\number\numexpr#1+543\relax}\edef#5{#2}\edef#6{#3}}
\def\bbl@cs@chinese@leap{885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
\def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,354,384,413,443,472,501,531,560,590,620,649,679,709,738,768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
\def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,354,384,413,443,472,501,531,560,590,620,649,679,709,738,768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
\end{macrocode}

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

14.1 Not renaming hyphen.tex

As Don Knuth has declared that the filename hyphen.tex may only be used to designate his version of the American English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based \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 hyphen.tex or whatever they like, but they mustn't diddle with hyphen.tex (or plain.tex except to reload additional fonts).

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

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

168

诱人\input\ #1 
168  \
\let\input\a
168  \
\a hyphen.cfg

168
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 \LaTeX \TeX 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 \texttt{\textbackslash babeloptionstrings} and \texttt{\textbackslash babeloptionmath} are provided, which can be defined before loading babel. \texttt{\textbackslash BabelModifiers} can be set too (but not sure it works).

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.

### 14.3 General tools

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

\long\def\@firstofone#1{#1} \long\def\@firstoftwo#1#2{#1} \long\def\@secondoftwo#1#2{#2} \def\@nnil\@nil \def\@gobbletwo#1#2{} \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}} \def\@star@or@long#1{\@ifstar{\let\l@ngrel@x\relax#1}} \let\l@ngrel@x\relax \def\@car#1#2\@nil{#1} \def\@cdr#1#2\@nil{#2} \let\@typeset@protect\relax \let\protected@edef\edef \long\def\@gobble#1{} \edef\@backslashchar\expandafter\@gobble\string\\ \def\strip@prefix#1>{\def\Strip@Prefix#1>{} \def\strip@prefix#1>[}{}{\let\strip@prefix\undefined} \def\strip@prefix#1|>{\def\Strip@Prefix#1|>{} {\let\strip@prefix\undefined} \def\strip@prefix#1|}[}{}{\let\strip@prefix\undefined}
\toks@\expandafter{#1#2}\
\xdef#1{\the\toks@}}
\def\@namedef#1{\expandafter\def\csname #1\endcsname}
\def\@nameuse#1{\csname #1\endcsname}
\def\@ifundefined#1{\expandafter\ifx\csname#1\endcsname\relax\expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
\def\@expandtwoargs#1#2#3{\edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
\def\zap@space#1 #2{#1\ifx#2\@empty\else\expandafter\zap@space\fi#2}
\let\bbl@trace\@gobble
\def\bbl@error#1{\begingroup\catcode`\=0 \catcode`\=12 \catcode`\%=14\input errbabel.def\bbl@error{#1}}
\def\bbl@warning#1{\begingroup\newlinechar=`\^^J\def\{\&\bbl@warning\message{\#1}\endgroup}
\let\bbl@infowarn\bbl@warning
\def\bbl@info#1{\begingroup\newlinechar=`\^^J\def\{\&\wlog{#1}\endgroup}
LaTeX has the command \@onlypreamble which adds commands to a list of commands that are no longer needed after \begin{document}.
\ifx\@preamblecmds\@undefined\def\@preamblecmds{}\fi
\def\@onlypreamble#1{\expandafter\gdef\expandafter\@preamblecmds\expandafter{\@preamblecmds\do#1}}
\global\let\do\noexpand
\ifx\@begindocumenthook\@undefined\def\@begindocumenthook{}
\fi
\@onlypreamble\@begindocumenthook
\def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

Mimic \LaTeX's \AtBeginDocument; for this to work the user needs to add \begindocument to his file.
\def\begindocument{}\begindocument
\@begindocumenthook\global\let\@begindocumenthook\@undefined
\def\do#1{\global\let#1\@undefined}\@preamblecmds\global\let\do\noexpand
\ifx\@begindocumenthook\@undefined\def\@begindocumenthook{}
\fi
\@onlypreamble\@begindocumenthook
\def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

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

\begin{verbatim}
\catcode`\&=\z@\ifx&\texttt{if@filesw}\@undefined\expandafter\let\csname if@filesw\endcsname\csname iffalse\endcsname\fi\catcode`\&=4
\end{verbatim}

Mimic \LaTeX’s commands to define control sequences.

\begin{verbatim}
\newcommand{\@star@or@long}{new@command}
\newcommand{\@testopt{\ifx\@newcommand}}
\newcommand{\@newcommand#1}{\@testopt{\@newcommand#1}0}
\newcommand{\@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{\@protected@testopt\csname\string#1\endcsname{\#3}\expandafter\@yargdef\csname\string#1\endcsname\tw@{\#2}{\#4}}\def\providecommand{\@star@or@long}{provide@command}
\provide@command#1{\begingroup\escapechar\m@ne\xdef\@gtempa{\string#1}\endgroup\expandafter\@ifundefined\@gtempa{\def\reserved@a{\new@command#1}}{\let\reserved@a\relax\def\reserved@a{\new@command\reserved@a}}\reserved@a}
\DeclareRobustCommand{\@star@or@long}{\declare@robustcommand}
\declare@robustcommand#1{\edef\reserved@a{\string#1}\def\reserved@b{#1}\edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}\edef#1{\ifx\reserved@a\reserved@b\noexpand\x@protect\noexpand#1}}
\end{verbatim}
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@fakedd 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{\}

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 2ε 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 (\count10).

\ifx\bye\@undefined
\advance\count10 by -2\relax
\fi
\ifx@ifnextchar\@undefined
\def@ifnextchar#1#2#3{\let\reserved@d=#1\def\reserved@a{#2}\def\reserved@b{#3}\futurelet\@let@token@ifnch}
\def@ifnch{\ifx\@let@token\@sptoken

172
14.4 Encoding related macros

Code from ltoutenc.dtx, adapted for use in the plain TeX environment.
Currently we only use the \ipa 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 plain \LaTeX.

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

And a few more “dummy” definitions.
15 Acknowledgements

I would like to thank all who volunteered as β-testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spith 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