Babel

Localization and internationalization

Unicode
\TeX
pdf\TeX
Lua\TeX
Xe\TeX

Code
Version 3.99
2023/12/10

Javier Bezos
Current maintainer

Johannes L. Braams
Original author
10 Support for interchar
  10.1 Layout ........................................ 105
  10.2 8-bit TeX ...................................... 107
  10.3 LuaTeX ......................................... 107
  10.4 Southeast Asian scripts ......................... 113
  10.5 CJK line breaking .............................. 115
  10.6 Arabic justification ............................ 117
  10.7 Common stuff .................................. 121
  10.8 Automatic fonts and ids switching .............. 121
  10.9 Bidi ........................................... 128
  10.10 Layout ........................................ 130
  10.11 Lua: transforms .............................. 137
  10.12 Lua: Auto bidi with basic and basic-r .......... 146

11 Data for CJK ...................................... 156

12 The 'nil' language ................................ 157

13 Calendars ......................................... 158
  13.1 Islamic ........................................ 158
  13.2 Hebrew ......................................... 159
  13.3 Persian ......................................... 163
  13.4 Coptic and Ethiopic ............................ 164
  13.5 Buddhist ....................................... 165

14 Support for Plain TeX (plain.def) ................. 166
  14.1 Not renaming hyphen.tex ........................ 166
  14.2 Emulating some EłyX features .................. 167
  14.3 General tools .................................. 167
  14.4 Encoding related macros ....................... 171

15 Acknowledgements ................................ 174
The babel package is being developed incrementally, which means parts of the code are under development and therefore incomplete. Only documented features are considered complete. In other words, use babel in real documents only as documented (except, of course, if you want to explore and test them).

1 Identification and loading of required files

Code documentation is still under revision. The babel package after unpacking consists of the following files: 

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

There are some additional tex, def and lua files

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

2 locale directory

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

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

3 Tools

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

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

```latex
\langle\langle version=3.99\rangle\rangle
\langle\langle date=2023/12/10\rangle\rangle

\begin{verbatim}
1 \langle\langle Basic macros\rangle\rangle
2 \langle\langle\langle Basic macros\rangle\rangle
3 \langle\langle\langle Basic macros\rangle\rangle
4 \langle\langle\langle Basic macros\rangle\rangle
5 \langle\langle Basic macros\rangle\rangle
6 \langle\langle Basic macros\rangle\rangle
7 \langle\langle Basic macros\rangle\rangle
8 \langle\langle Basic macros\rangle\rangle
9 \langle\langle Basic macros\rangle\rangle
10 \langle\langle Basic macros\rangle\rangle
11 \langle\langle Basic macros\rangle\rangle
12 \langle\langle Basic macros\rangle\rangle
13 \langle\langle Basic macros\rangle\rangle
14 \langle\langle Basic macros\rangle\rangle
15 \langle\langle Basic macros\rangle\rangle
16 \langle\langle Basic macros\rangle\rangle
17 \langle\langle Basic macros\rangle\rangle
```

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.

```
\def\bbl@add@list#1#2{\edef#1{\bbl@ifunset{\bbl@stripslash#1}{}{\ifx#1\@empty\else#1,\fi}#2}}
```

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\else\fi statement\footnote{This code is based on code presented in TUGboat vol. 12, no2, June 1991 in “An expansion Power Lemma” by Sonja Maus.}. These macros will break if another \if\else\fi statement appears in one of the arguments and it is not enclosed in braces.

```
\def\bbl@afterelse#1\else#2\fi{\fi#1}
\def\bbl@afterfi#1\fi{\fi#1}
```

Now, just syntactical sugar, but it makes partial expansion of some code a lot more simple and readable. Here \noexpand means \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.

```
\def\bbl@tempa#1{\long\def\bbl@trim##1##2{\futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}\def\bbl@trim@c{\ifx\bbl@trim@a\@sptoken\expandafter\bbl@trim@b\else\expandafter\bbl@trim@b\expandafter#1\fi}\long\def\bbl@trim@b#1##1\@nil\{\bbl@trim@i##1\}}\long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}\long\def\bbl@trim@def#1{\bbl@trim{\def#1}}}
```

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

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

\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#5\@nil#6\@secondoftwo#7\@firstoftwo#8\fi\else\expandafter\@secondoftwo\fi
\def\bbl@ifset#1#2#3{%\bbl@ifunset{#1}{#3}{\bbl@exp{\@empty}{#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{%\bbl@kvcmd##1##2##3{#2}\bbl@fornext#1,\@nil,}
\def\bbl@fornext#1,{%\ifx\@nil#1\relax\else\bbl@ifblank{#1}{}{\bbl@forkv@eq#1=\@empty=\@nil{#1}}\expandafter\bbl@fornext\fi}
\def\bbl@forkv@eq#1=#2=#3\@nil#4{%\bbl@trim@def\bbl@forkv@a{#1}\bbl@trim\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@vforeach#1{%\bbl@forcmd##1{#2}\bbl@fornext#1,\@nil,}
\def\bbl@fornext#1,{%\ifx\@nil#1\relax\else\bbl@ifblank{#1}{}{\bbl@trim\bbl@forcmd{#1}}\expandafter\bbl@fornext\fi}

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

\def\bbl@replace#1#2#3{\toks@{}\def\bbl@replace@aux#1\@empty\@nil#2\@empty\@nil#3\@empty\@nil{#2}}

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

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

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

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

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

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

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

Now we make sure all required files are loaded. When the command \AtBeginDocument doesn't exist we assume that we are dealing with a plain-based format. In that case the file plain.def is needed (which also defines \AtBeginDocument, and therefore it is not loaded twice). We need the first part when the format is created, and \orig@dump is used as a flag. Otherwise, we need to use the second part, so \orig@dump is not defined (plain.def undefines it).

Check if the current version of switch.def has been previously loaded (mainly, hyphen.cfg). If not, load it now. We cannot load babel.def here because we first need to declare and process the package options.

### 3.2 The Package File (\LaTeX{}, babel.sty)

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

Start with some “private” debugging tool, and then define macros for errors.
This file also takes care of a number of compatibility issues with other packages and defines a few additional package options. Apart from all the language options below we also have a few options that influence the behavior of language definition files.

Many of the following options don't do anything themselves, they are just defined in order to make it possible for babel and language definition files to check if one of them was specified by the user. But first, include here the Basic macros defined above.

```latex
\ifpackagewith{babel}{silent}
  \let\bbl@info@\gobble
  \let\bbl@infolist@\gobble
  \let\bbl@warning@\gobble
\fi
```

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

```latex
\ifx\bbl@languages\undefined\else
  \begingroup
    \catcode`\^^I=12
    \ifpackagewith{babel}{showlanguages}{
      \def\bbl@elt#1#2#3#4{
        \wlog{#2^^I#1^^I#3^^I#4}}
      \wlog{<*languages>}
      \bbl@languages
      \wlog{</languages>}
    }{}
    \def\bbl@elt#1#2#3#4{
      \ifnum#2=\z@
        \gdef\bbl@nulllanguage{#1}{}
      \fi}
    \bbl@languages
  \fi
\endgroup
```

### 3.3 base

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

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

```latex
\bbl@trace{Defining option `base'}
\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

  \bbl@languages
}
```

```latex
\ProcessOptions
\global\expandafter\let\csname opt@babel.sty\endcsname\relax
\global\expandafter\let\csname ver@babel.sty\endcsname\relax
\global\let\if@ter@\@ifl@ter
\def\@ifl@ter#1#2#3#4#5{\global\let\if@ter@\@ifl@ter@}
\endinput}
```
3.4 key=value options and other general options

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.

\begin{verbatim}
\bbl@trace{key=value and another general options}
\bbl@csarg\let{tempa\expandafter}{\csname opt@babel.sty\endcsname}
\def\bbl@temb#1.#2{% Remove trailing dot
#1\ifx\@empty#2\else,\bbl@afterfi\bbl@temb#2\fi}%
\def\bbl@tempf#1=#2\@@{%
\bbl@csarg\edef{mod@#1}{\bbl@temb#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@{,provide=}{,#1}%
\ifin@
\edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@temb#2}%
\else
\in@{$modifiers$}{$#1$}% TODO. Allow spaces.
\ifin@
\edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
\else
\edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
\bbl@csarg\edef{mod@#1}{\bbl@temb#2}%
\fi%
\fi
\fi}
\let\bbl@tempc\@empty
\bbl@foreach\bbl@tempa{\bbl@temb#1.\@empty\@nnil}
\expandafter\let\csname opt@babel.sty\endcsname\bbl@tempc
\end{verbatim}

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

\begin{verbatim}
\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
\let\bbl@autoload@options\@empty
\DeclareOption{provide@=*}{\def\bbl@autoload@options{import}}
\newif\ifbbl@single
\DeclareOption{selectors=off}{\bbl@singletrue}
\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
\let\bbl@autoload@options\@empty
\DeclareOption{provide@=*}{\def\bbl@autoload@options{import}}
\newif\ifbbl@single
\DeclareOption{selectors=off}{\bbl@singletrue}
\end{verbatim}

Handling of package options is done in three passes. (I [JBL] am not very happy with the idea,
anyway.) The first one processes options which has been declared above or follow the syntax
<key>=<value>, the second one loads the requested languages, except the main one if set with the
key main, and the third one loads the latter. First, we “flag” valid keys with a nil value.

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

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

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

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

Now we finish the first pass (and start over).
\ProcessOptions*
\let\bbl@opt@provide\@empty
\ifx\bbl@opt@provide\@nil
  \let\bbl@opt@provide\@empty
  \%\% MOVE above
\else
  \chardef\bbl@iniflag\@ne
  \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
    \in@{,provide,}{,#1,}%
    \ifin@
    \def\bbl@opt@provide#2%}
    \bbl@replace\bbl@opt@provide{;}{,}%
  \fi
}\fi

\% 3.5 Conditional loading of shorthands

If there is no shorthands=<chars>, the original babel macros are left untouched, but if there is, these
macros are wrapped (in babel.def) to define only those given.
A bit of optimization: if there is no shorthands=, then \bbl@ifshorthand is always true, and it is
always false if shorthands is empty. Also, some code makes sense only with shorthands=....

\bbl@trace{Conditional loading of shorthands}
\def\bbl@sh@string#1{%
  \ifx#1\empty\else
    \ifx#1t\string~%
    \else\ifx#1c\string,%
    \else\string#1%
  \fi
}\fi
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.

\g@addto@macro\resetactivechars{\let\protect\noexpand}

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

\edef\bbl@opt@safe{BR}
\let\bbl@opt@safe\@empty % Pending of \cite

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

\begin{input}
\@secondoftwo
\fi
\endinput % Same line!
\ProvidesFile{babel.def} [% Make sure ProvidesFile is defined
\date v [Babel common definitions]
\AtBeginDocument % TODO. change test.
\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.

\endinput % <core>
\ProvidesFile{switch.def} [% Multiple languages
Plain\TeX\ version 3.0 provides the primitive \texttt{\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 \texttt{\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%
\begin{group}
\count@#1
\def\bbl@elt##1##2##3##4{%\ifnum\count@=##2\relax
\edef\bbl@tempa{\expandafter\@gobbletwo\string#1}\
\bbl@info{Hyphen rules for '\string\csname l@##1\endcsname\%\string\language\the\count@}. Reported}\
\fi}
\bbl@cs{languages}
\endgroup

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

\bbl@iflanguage#1%
\begin{group}
\bbl@tempa{\string\language\texttt{\count@}}
\endgroup
After a name has been ‘fixed’, the selectors will try to load the language. If even the fixed name is not defined, will load it on the fly, either based on its name, or if activated, its BCP47 code.

We first need a couple of macros for a simple BCP 47 look up. It also makes sure, with \bbl@bcpcase, casing is the correct one, so that sr-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.

\edef\bbl@bcpcase{#1#2#3#4@#5}{%
528 \def\bbl@provide@locale{%529  \ifx\bbl@provide@undefined530  \bbl@error{For a language to be defined on the fly 'base'\%
531  is not enough, and the whole package must be\%532  loaded. Either delete the 'base' option or\%533  request the languages explicitly}%534  \fi535  \let\bbl@auxname\languagename % Still necessary. TODO536  \bbl@ifunset{bbl@bcp@map@\languagename}{}% Move uplevel??537  \edef\languagename{\@nameuse{bbl@bcp@map@\languagename}}%538  \ifbbl@bcpallowed
539 \begin{verbatim}540  \expandafter\if\csname date\languagename\endcsname\relax
541  \ifx\bbl@bcp\relax% Returned by \bbl@bcplookup
542  \edef\languagename{\bbl@bcp@prefix\bbl@bcp}\edef\localename{\bbl@bcp@prefix\bbl@bcp}
543  \expandafter\if\csname date\languagename\endcsname\relax
544  \let\bbl@initoload\bbl@bcp
545  \bbl@exp{\\babelprovide{\bbl@autoload@bcpoptions}{\languagename}}%546  \let\bbl@initoload\relax
547 \end{verbatim}548  \fi
549 \fi550 \expandafter\if\csname date\languagename\endcsname\relax
551 \IfFileExists{babel-\languagename.tex}{\bbl@exp{\\babelprovide{\bbl@autoload@options}{\languagename}}}{}
552 \fi
553 \fi
554 \fi
555 \expandafter\if\csname date\languagename\endcsname\relax
556 \IfFileExists{babel-\languagename.tex}{\bbl@exp{\\babelprovide{\bbl@autoload@options}{\languagename}}}{}
557 \fi
558 \fi
559 \fi
560 \end{verbatim}

\iflanguage

Users might want to test (in a private package for instance) which language is currently active. For this we provide a test macro, \iflanguage, that has three arguments. It checks whether the first argument is a known language. If so, it compares the first argument with the value of \language. Then, depending on the result of the comparison, it executes either the second or the third argument. 4.1 Selecting the language

\selectlanguage

The macro \selectlanguage checks whether the language is already defined before it performs its actual task, which is to update \language and activate language-specific definitions. 561 \def\iflanguage#1{%562  \bbl@iflanguage{#1}{%563  \ifnum\csname l@#1\endcsname=\language564  \expandafter\@firstoftwo
565  \else
566  \expandafter\@secondoftwo
567  \fi}}

\begin{verbatim}
\end{verbatim}

\selectlanguage

The macro \selectlanguage checks whether the language is already defined before it performs its actual task, which is to update \language and activate language-specific definitions. 568 \let\bbl@select@type@z@569 \edef\selectlanguage{%570 \noexpand\protect
571 \expandafter\noexpand\csname selectlanguage \endcsname
572 \end{verbatim}

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

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

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

573 \let\xstring\string


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

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

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

\bbl@push@language \bbl@language@stack

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

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

\bbl@push@language\bbl@language@stack

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

\bbl@pop@lang\bbl@language@stack

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

\bbl@id@assign 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).

\bbl@id@assign % No real need for a new counter
\bbl@id@assign % \bbl@id@assign\languagename\% 
\chardef\localeid\z@ % localeid... will be reserved for hyphenation patterns (so that two locales can share the same rules).
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).
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, we also redefine the macro \noextras which is used to suppress extra spaces in the output. Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with \select@language, 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 the values of \lefthyphenmin and \righthyphenmin are defined. If it is not, we set default values (2 and 3), otherwise the values in \languagename will be used.

No text is supposed to be added with switching captions and date, so we remove any spurious spaces with \bb@bsphack and \bb@esphack.
otherlanguage (em.) The otherlanguage environment can be used as an alternative to using the \selectlanguage declarative command. When you are typesetting a document which mixes left-to-right and right-to-left typesetting you have to use this environment in order to let things work as you expect them to. The \ignorespaces command is necessary to hide the environment when it is entered in horizontal mode.

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

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

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\langle lang\rangle command doesn't make any \global changes. The coding is very similar to part of \selectlanguage. \bbl@beforeforeign is a trick to fix a bug in bidi texts. \foreignlanguage is supposed to be a 'text' command, and therefore it must emit a \leavevmode, but it does not, and therefore the indent is placed on the opposite margin. For backward compatibility, however, it is done only if a right-to-left script is requested; otherwise, it is no-op.

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

(3.11) Also experimental are the hook foreign and foreign*. With them you can redefine \BabelText which by default does nothing. Its behavior is not well defined yet. So, use it in horizontal mode only if you do not want surprises.
In other words, at the beginning of a paragraph \texttt{\foreignlanguage} enters into hmode with the surrounding lang, and with \texttt{\foreignlanguage*} with the new lang.

\providecommand\bbl@beforeforeign{}
\edef\foreignlanguage{\noexpand\protect\expandafter\noexpand\csname foreignlanguage \endcsname}
\expandafter\def\csname foreignlanguage \endcsname{\@ifstar\bbl@foreign@s\bbl@foreign@x}
\providecommand\bbl@foreign@x[3][{\begingroup\def\bbl@selectorname{foreign}\def\bbl@select@opts{#1}\let\BabelText@firstofone\bbl@beforeforeign\foreignlanguage{#2}\bbl@usehooks{foreign}{}\BabelText{#3} now in horizontal mode!\endgroup}
\def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par\begingroup\par\def\bbl@selectorname{foreign*}\let\bbl@select@opts\@empty\let\BabelText@firstofone\foreignlanguage{#1}\bbl@usehooks{foreign*}{}\bbl@dirparastext\BabelText{#2} still in vertical mode!\par\endgroup}
\foreign@language This macro does the work for \texttt{\foreignlanguage} and the \texttt{\foreignlanguage*} environment. First we need to store the name of the language and check that it is a known language. Then it just calls \texttt{\bbl@switch}.

\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\expandafter\bbl@switch\expandafter\languagename}}

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

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

\texttt{\bbl@patterns} This macro selects the hyphenation patterns by changing the \texttt{\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 \texttt{language} \texttt{\lccode}'s has been set, too). \texttt{\bbl@hyphenation@} is set to relax until the very first \texttt{\babelhyphenation}, so do nothing with this value. If the exceptions for a language (by its number, not its name, so that \texttt{:ENC} is
taken into account) has been set, then use \hyphenation with both global and language exceptions and empty the latter to mark they must not be set again.

\let\bbl@hyphlist\@empty
\let\bbl@hyphenation@\relax
\let\bbl@pttnlist\@empty
\let\bbl@patterns@\relax
\let\bbl@hymapsel=\@cclv
\def\bbl@patterns#1{%
  \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
  \csname l@#1\endcsname
  \edef\bbl@tempa{#1}\
  \else
  \csname l@#1:\f@encoding\endcsname
  \edef\bbl@tempa{#1:\f@encoding}\
  \fi
  \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
% > luatex
  \@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
  \begingroup
  \bbl@xin@{,\number\language,}{,\bbl@hyphlist}\
  \ifin@\else
  \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
  \hyphenation{%
    \bbl@hyphenation@%
    \@ifundefined{bbl@hyphenation@#1}\
    \@empty
    \space\csname bbl@hyphenation@#1\endcsname}%
  \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
  \fi
  \endgroup}}

\hyphenrules 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*.

\def\hyphenrules#1{%
  \edef\bbl@tempf{#1}%
  \bbl@fixname\bbl@tempf
  \bbl@iflanguage\bbl@tempf{%
    \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
    \ifx\languageshorthands\@undefined\else
    \languageshorthands{none}%
    \expandafter\expandafter\expandafter\set@hyphenmins
    \csname\bbl@tempf hyphenmins\endcsname\relax
    \@empty
    \space\csname bbl@hyphenation@#1\endcsname}%
    \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
  \fi
  \endgroup}}

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

\def\providehyphenmins#1#2{%
  \edef\bbl@tempf{#1}%
  \bbl@fixname\bbl@tempf%
  \expandafter\expandafter\expandafter\set@hyphenmins
  \csname\bbl@tempf hyphenmins\endcsname\relax
  \set@hyphenmins{tw@thr@@}%
  \else
  \expandafter\expandafter\expandafter\set@hyphenmins
  \csname bbl@hyphlist\number\language,}%
  \endgroup}}
\begin{lrbox}{\@tempboxb}
\end{lrbox}
\endinput
\ProvidesLanguage

The identification code for each file is something that was introduced in \TeX\2ε. When the command \ProvidesFile does not exist, a dummy definition is provided temporarily. For use in the language definition file the command \ProvidesLanguage is defined by babel.

Depending on the format, i.e., on if the former is defined, we use a similar definition or not.

\ifdef\ProvidesFile\@undefined
\def\ProvidesLanguage#1[#2 #3 #4]{%
  \wlog{Language: #1 #4 #3 <#2>}%
}\
\else
\def\ProvidesLanguage#1{%
  \begingroup
  \catcode`\ 10 \%
  \@makeother/\%
  \@ifnextchar[\]{{\@provideslanguage{#1}}}{{\@provideslanguage{#1}[]}}
}\
\def\@provideslanguage#1[#2]{%
  \wlog{Language: #1 #2}%
  \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
}\
\endgroup
}\fi

\originalTeX

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

\ifdef\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.

\ifdef\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}{Find an armchair, sit down and wait}}
  \let\uselocale\setlocale
  \let\locale\setlocale
  \let\selectlocale\setlocale
  \let\textlocale\setlocale
  \let\textlanguage\setlocale
  \let\languagetext\setlocale

4.2 Errors

\nolanerr
\nopatterns

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

\nooterr

When the package was loaded without options not everything will work as expected. An error message is issued in that case. When the format knows about \PackageError it must be \TeX\2ε, so we can safely use its error handling interface. Otherwise we’ll have to ‘keep it simple’. Infos are not written to the console, but on the other hand many people think warnings are errors, so a further message type is defined: an important info which is sent to the console.

\edef\bbl@nulllanguage{\string\language=0}
  \def\bbl@nocaption\protect\bbl@nocaption@i
  \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
  \global\@namedef{#2}{\textbf{?#1?}}%
  \@nameuse{#2}%
\edef\bbl@tempa{#1}\%  
\bbl@sreplace\bbl@tempa{name}{}\%  
\bbl@warning{  
(\textbackslash@backslashchar#1 not set for '{\languagename}'. Please,\%  
define it after the language has been loaded\%  
typically in the preamble) with:\%  
\string\setlocalecaption{{\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\%  
{You haven't defined the language '{#1}' yet.\%  
Perhaps you misspelled it or your installation\%  
is not complete}\%  
{Your command will be ignored, type <return> to proceed}}\%  
\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 {'\bbl@nulllanguage'} 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\fi\%  
\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\%  
\chardef\l@english\z@\%  
\addto\protect It takes two arguments, a ⟨control sequence⟩ and TeX-code to be added to the ⟨control sequence⟩.
If the \textit{control sequence} has not been defined before it is defined now. The control sequence could also expand to \texttt{\relax}, in which case a circular definition results. The net result is a stack overflow. Note there is an inconsistency, because the assignment in the last branch is global.

\begin{verbatim}
def\addto#1#2{\
  \ifx#1\@undefined\
    \def#1{#2}\
  \else\
    \ifx#1\relax\
      \def#1{#2}\
    \else\
      {\toks@\expandafter{#1#2}}\
    \fi\
  \fi\
}\end{verbatim}

The macro \texttt{\initiate@active@char} below takes all the necessary actions to make its argument a shorthand character. The real work is performed once for each character. But first we define a little tool.

\begin{verbatim}
def\bbl@withactive#1#2{\
  \begingroup\
  \lccode`~=`#2\relax\
  \lowercase{\endgroup#1~}}\end{verbatim}

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

\begin{verbatim}
def\bbl@redefine#1{\
  \edef\bbl@tempa{\bbl@stripslash#1}\
  \expandafter\let\csname org@\bbl@tempa\endcsname#1\
  \expandafter\def\csname\bbl@tempa\endcsname}\end{verbatim}

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

\begin{verbatim}
def\bbl@redefine@long#1{\
  \edef\bbl@tempa{\bbl@stripslash#1}\
  \bbl@ifunset{\bbl@tempa\space}\
  {\expandafter\let\csname org@\bbl@tempa\endcsname#1\
    \bbl@exp{\def\#1{\protect<\bbl@tempa\space}}}\
  {\bbl@exp{\let<\org@\bbl@tempa<\bbl@tempa\space}}}\
  \@namedef{\bbl@tempa\space}}\end{verbatim}

\texttt{\bbl@redefinerobust} For commands that are redefined, but which \textit{might} be robust we need a slightly more intelligent macro. A robust command \texttt{foo} is defined to expand to \texttt{\protect\foo}. So it is necessary to check whether \texttt{\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 \texttt{\foo}.

\begin{verbatim}
def\bbl@redefinerobust#1{\
  \edef\bbl@tempa{\bbl@stripslash#1}\
  \bbl@ifunset{\bbl@tempa\space}\
  {\expandafter\let\csname org@\bbl@tempa\endcsname#1\
    \bbl@exp{\def\#1{\protect<\bbl@tempa\space}}}\
  {\bbl@exp{\let<\org@\bbl@tempa<\bbl@tempa\space}}}\
  \@namedef{\bbl@tempa\space}}\end{verbatim}

\subsection{Hooks}

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

\begin{verbatim}
def\bbl@usehooks{\texttt{\enableBabelHook[#2]}}\end{verbatim}
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 \texttt{hyphen.cfg} are also loaded (just in case you need them for some reason).

To define keywords, arguments in \texttt{\textbackslash def}\texttt{\textbackslash tempa\#1,\#3=}\texttt{\textbackslash empty}{\texttt{\textbackslash def}\texttt{\textbackslash tempb\#2}}

\begin{verbatim}
\def\bbl@tempa##1,#3=##2,\@empty{
\def\bbl@tempb{##2}}
\expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
\bbl@ifunset{bbl@ev@#2@#3@#1}{\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
\bbl@csarg\let{ev@#2@#3@#1}\relax
\bbl@csarg\newcommand{ev@#2@#3@#1}{\bbl@tempb}
\end{verbatim}

\newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
\newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
\def\bbl@usehooks{\bbl@usehooks@lang\languagename}
\def\bbl@usehooks@lang#1#2#3{% Test for Plain
\ifx\UseHook\@undefined\else\UseHook{babel/*/#2}\fi
\def\bbl@elth##1{\bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@}#3}}%
\bbl@cs{ev@#2@}%
\ifx\languagename\@undefined\else% Test required for Plain (?)
\ifx\UseHook\@undefined\else\UseHook{babel/#1/#2}\fi
\def\bbl@elth##1{\bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#1}#3}}%
\bbl@cs{ev@#2@#1}%
\fi}

To define \texttt{\textbackslash def}\texttt{\textbackslash 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 (?)
\def\bbl@tempa#1=#2\@@{\NewHook{babel/#1}}
\bbl@foreach\bbl@evargs{\bbl@tempa#1\@@}
\fi

The user command just parses the optional argument and creates a new macro named \texttt{\bbl@e@⟨language⟩}. 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@⟨language⟩} contains \texttt{\bbl@ensure{⟨include⟩}{⟨exclude⟩}{⟨fontenc⟩}}, 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 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.

\begin{verbatim}
\bbl@trace{Defining babelensure}
\newcommand\babelensure[2][2]{% 
\AddBabelHook{babel-ensure}{afterextras}{% 
\ifcase\bbl@select@type
\bbl@cl{e}% 
\fi}%
\begingroup
\let\bbl@ens@include\@empty
\let\bbl@ens@exclude\@empty
\def\bbl@ens@fontenc{\relax}%
\def\bbl@tempb##1{\ifx\@empty##1\else
\noexpand##1\expandafter\bbl@tempb\fi}%
\edef\bbl@tempa{\bbl@tempb#1\@empty}%
\def\bbl@tempb##1=##2\@@{\@namedef{bbl@ens@##1}{##2}}%
\bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
\def\bbl@tempc{\bbl@ensure}%
\expandafter\bbl@add\expandafter\bbl@tempc\expandafter{% 
\expandafter{\bbl@ens@include}}%
\expandafter\bbl@add\expandafter\bbl@tempc\expandafter{% 
\expandafter{\bbl@ens@exclude}}%
\expandafter\bbl@add\expandafter\bbl@tempc\expandafter{% 
\expandafter{\bbl@ens@fontenc}{\relax}}%
\edef\bbl@tempp#1{\ifx\empty#1\else
\noexpand#1\expandafter\bbl@tempp\fi}%
\edef\bbl@tempp#1=##2\@@{\@namedef{bbl@ens@##1}{##2}}%
\bbl@foreach\bbl@tempp{\bbl@tempp#1\@@}%
\expandafter\bbl@add\expandafter\bbl@tempp\expandafter{% 
\expandafter{\bbl@ens@include}}%
\expandafter\bbl@add\expandafter\bbl@tempp\expandafter{% 
\expandafter{\bbl@ens@exclude}}%
\endgroup
\end{verbatim}
4.4 Setting up language files

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

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

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

Now we check whether we should perhaps stop the processing of this file. To do this we first need to check whether the second argument that is passed to \LdfInit is a control sequence. We do that by looking at the first token after passing \#2 through string. When it is equal to @\backslashchar we are dealing with a control sequence which we can compare with \@undefined. If so, we call \ldf@quit to set the main language, restore the category code of the @-sign and call
When \#2 was not a control sequence we construct one and compare it with \relax.

Finally we check \originalTeX.

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

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

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

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

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

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

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

\begin{verbatim}
1149 \onlypreamble\LdfInit
1150 \onlypreamble\ldf@quit
1151 \onlypreamble\ldf@finish
\end{verbatim}

28
This command should be used in the various language definition files. It stores its argument in \bbl@main@language; to be used to switch to the correct language at the beginning of the document.

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

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

\def\bbl@beforestart{%\def@nolanerr##1{%\bbl@warning{Undefined language '##1' in aux.\Reported}}%\bbl@usehooks{beforestart}{}%\global\let\bbl@beforestart\relax}\AtBeginDocument{%{\@nameuse{bbl@beforestart}}% Group!
\if@filesw\providecommand\babel@aux[2]{}%\immediate\write\@mainaux{\string\providecommand\string\babel@aux[2]{}}%\immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%\fi\expandafter\selectlanguage\expandafter{\bbl@main@language}⟨\-core⟩\ifx\bbl@normalsf\@empty\ifnum\sfcode`.\@m\let\normalsfcodes\frenchspacing\else\let\normalsfcodes\nonfrenchspacing\fi\else\let\normalsfcodes\bbl@normalsf\fi\ifbbl@single % must go after the line above.\renewcommand\selectlanguage[1]{}%\renewcommand\foreignlanguage[2]{#2}%\global\let\babel@aux\@gobbletwo % Also as flag\fi\-core\AddToHook{begindocument/before}{%\let\bbl@normalsf\normalsfcodes\let\normalsfcodes\relax} % Hack, to delay the setting\+core\ifcase\bbl@engine\or\AtBeginDocument{\pagedir\bodydir} % TODO - a better place\fi

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

\bbl@trace{Shorthands}
\def\bbl@add@special#1{% 1:a macro like ", ?, etc.
\bbl@add\dospecials{do#1}% test @sanitize = \relax, for back. compat.
\bbl@ifunset{@sanitize}{}{%139\bbl@add\@sanitize{\@makeother#1}}%
\ifx\nfss@catcodes\@undefined\else % TODO - same for above
\begingroup
\catcode`#1\active
\nfss@catcodes
\ifnum\catcode`#1=\active
\endgroup
\bbl@add\nfss@catcodes{%139\@makeother#1}%
\else
\endgroup
\fi
\fi}
\bbl@remove@special
The companion of the former macro is \bbl@remove@special. It removes a character from the set macros \dospecials and \@sanitize, but it is not used at all in the babel core.
\def\bbl@remove@special#1{%
\begingroup
\def\x##1##2{\ifnum`#1=`##2
oexpand\@empty
\else
oexpand##1
oexpand##2
\fi}%
\def\do{\x\do}%
\def\@makeother{\x\@makeother}%
\edef\x{\endgroup
\def
\noexpand\dospecials{\dospecials}%
\expandafter\ifx\csname @sanitize\endcsname\relax\else
\def\noexpand\@sanitize{\@sanitize}%
\fi}%
\x}
\initiate@active@char
A language definition file can call this macro to make a character active. This macro takes one argument, the character that is to be made active. When the character was already active this macro does nothing. Otherwise, this macro defines the control sequence \normal@char{char} to expand to the character in its ‘normal state’ and it defines the active character to expand to \normal@char{char} by default (\char being the character to be made active). Later its definition can be changed to expand to \active@char{char} by calling \bbl@activate{char}.

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

The following macro is used to define shorthands in the three levels. It takes 4 arguments: the (string'ed) character, \langle level\rangle@group, \langle level\rangle@active and \langle next-level\rangle@active (except in system).
\def\bbl@active@def#1#2#3#4{% 1228
\@namedef{#3#1}{%
\expandafter\ifx\csname#2\endcsname#1\relax
\bbl@afterelse\bbl@sh@select#2#1{#3#arg#1}{#4#1}%
\else
\bbl@afterfi\csname#2\endcsname
\fi}%
\bbl@afterfi%

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).

\begin{verbatim}
\def\@initiate@active@char#1{%  \ifx#1\@undefined  \bbl@csarg\def{oridef@@#2}#1%  \bbl@csarg\edef{oridef@#2}{%  \let
oexpand#1%  \expandafter
oexpand\csname bbl@oridef@@#2\endcsname}%  \else  \expandafter\let\csname normal@char#2\endcsname#3%\fi
\end{verbatim}

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

\begin{verbatim}
\ifx\#3\relax  \expandafter\let\csname normal@char#2\endcsname#3%\else  \expandafter\let\csname normal@char#2\endcsname#3%\fi
\end{verbatim}

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).

\begin{verbatim}
\bbl@restoreactive{#2}%
\AtBeginDocument{%  \ifcatcode`#2\active  \else  \immediate\write\@mainaux{\catcode`\string#2\active}%  \fi
\end{verbatim}

Now we have set \normal@char, we must define \active@char, to be executed when the character is activated. We define the first level expansion of \active@char to check the
status of the @safe@actives flag. If it is set to true we expand to the ‘normal’ version of this character; otherwise we call \user@active(char) to start the search of a definition in the user, language and system levels (or eventually normal@char(char)).

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

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

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

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

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

The next level of the code checks whether a user has defined a shorthand for himself with this character. First we check for a single character shorthand. If that doesn’t exist we check for a shorthand with an argument.

\bbl@active@def#2\user@group{user@active}{language@active}\
\bbl@active@def#2\language@group{language@active}{system@active}\
\bbl@active@def#2\system@group{system@active}{normal@char}

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

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

Finally, a couple of special cases are taken care of. (1) If we are making the right quote (’’) active we need to change \pr@m@s as well. Also, make sure that a single ‘ ‘ in math mode ‘does the right thing’. (2) If we are using the caret (^) as a shorthand character special care should be taken to make sure math still works. Therefore an extra level of expansion is introduced with a check for math mode on the upper level.

\if\string'#2%
  \let\prim@s\bbl@prim@s
  \let\active@math@prime@#1%
  \fi
\bbl@usehooks{initiateactive}{(#1){#2}{#3}}
The following package options control the behavior of shorthands in math mode.

\begin{verbatim}
\DeclareOption{math=active}{}
\DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
\end{verbatim}

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.

\begin{verbatim}
\ifpackagewith{babel}{KeepShorthandsActive}%
\let\bbl@restoreactive\@gobble%
\else
\bbl@exp{\AfterBabelLanguage\CurrentOption{\catcode`#1=\the\catcode`#1\relax}\AtEndOfPackage{\catcode`#1=\the\catcode`#1\relax}}%
\AtEndOfPackage{\let\bbl@restoreactive\@gobble}}
\end{verbatim}

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

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

\begin{verbatim}
\begingroup
\bbl@ifunset{ifincsname}% TODO. Ugly. Correct? Only Plain?
{\def\active@prefix#1{%
\ifx\protect\@typeset@protect
\else
\ifx\protect\@unexpandable@protect
\noexpand#1%
\else
\protect#1%
\fi
\expandafter\expandafter\expandafter\@gobble
\fi}
{\def\active@prefix#1{%
\ifincsname
\string#1%
\expandafter\expandafter\expandafter\@gobble
\fi}
{\def\active@prefix#1{%
\expandafter\expandafter\expandafter\@gobble
\else
\ifx\protect\@typeset@protect
\else
\ifx\protect\@unexpandable@protect
\noexpand#1%
\else
\protect#1%
\fi
\expandafter\expandafter\expandafter\@gobble
\fi
\fi
\fi}
\endgroup
\end{verbatim}
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@actives true), something like "13"_3 becomes "12"_3 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@actives false).

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

Both macros take one argument, like \initiate@active@char. The macro is used to change the definition of an active character to expand to \active@char\{\char\} in the case of \bbl@activate, or \normal@char\{\char\} in the case of \bbl@deactivate.

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

The command \declare@shorthand is used to declare a shorthand on a certain level. It takes three arguments:

1. a name for the collection of shorthands, i.e. ‘system’, or ‘dutch’;
2. the character (sequence) that makes up the shorthand, i.e. ~ or “a;
3. the code to be executed when the shorthand is encountered.

The auxiliary macro \bbl@expdf 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.
\textormath Some of the shorthands that will be declared by the language definition files have to be usable in both text and math mode. To achieve this the helper macro \textormath is provided.

\def\textormath{\ifmmode\expandafter\@secondoftwo\else\expandafter\@firstoftwo\fi}

\user@group \language@group \system@group

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

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

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

\def\useshorthands{%
  \@ifstar\@bbl@usesh@s\bbl@usesh@x{}}
\def\@bbl@usesh@s#1{%
  \bbl@activate{#1}}
\def\@bbl@usesh@x#1#2{%
  \bbl@error
  \\textbf{I can't declare a shorthand turned off ($\text{#2}$)}
  \\textbf{(Sorry, but you can't use shorthands which have been turned off in the package options)}}

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

\def\user@language@group{user@\language@group}
\def\bbl@set@user@generic#1{%
\languageshorthand  A user level command to change the language from which shorthands are used. Unfortunately, `babel` currently does not keep track of defined groups, and therefore there is no way to catch a possible change in casing to fix it in the same way languages names are fixed. [TODO].

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

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

\def\aliasshorthand#1#2{\bbl@ifshorthand{#2}{\expandafter\ifx\csname active@char\string#2\endcsname\relax
\if\document@notprerr
\@notshorthand{#2}
\else
\initiate@active@char{#2}
\bbl@ccarg\let{active@char\string#2}{active@char\string#1}
\bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}
\bbl@activate{#2}
\fi
\fi\}
\bbl@error{\The character '\string #1' should be made a shorthand character;\%
add the command \string\useshorthands\string{"\string} to
the preamble.\%
I will ignore your instruction}%
{You may proceed, but expect unexpected results}}

\@notshorthand

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

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

\shorthandoff  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}
\def\bbl@switch@sh#1#2{%
  \ifx#2\@nnil\else
    \bbl@ifunset{bbl@active@\string#2}\
      {\bbl@error
        {I can't switch '\string#2' on or off--not a shorthand}%
        {This character is not a shorthand. Maybe you made\%
          a typing mistake? I will ignore your instruction.}}}%
  {\ifcase#1% off, on, off*
    \catcode`#212\relax
    \or
    \catcode`#2\active
    \bbl@ifunset{bbl@shdef@\string#2}{}%
    \bbl@withactive{\let\csname bbl@shdef@\string#2\endcsname}{shdef@\string#2}\relax
  }%
  \ifcase\bbl@activated\or
    \bbl@activate{#2}%
  \else
    \bbl@deactivate{#2}%
  \fi
  \bbl@afterfi\bbl@switch@sh#1%
}\fi
\end{verbatim}

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

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

You may want to test if a character is a shorthand. Note it does not test whether the shorthand is on
One of the internal macros that are involved in substituting \prime for each right quote in math mode is \prims. 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{\prime\futurelet\@let@token\bbl@pr@m@s}
\def\bbl@if@primes#1#2{\ifx#1\@let@token\expandafter\@firstoftwo\else\ifx#2\@let@token\bbl@afterelse\expandafter\@firstoftwo\else\bbl@afterfi\expandafter\@secondoftwo\fi\fi}
\begingroup
\catcode`^=7 \catcode`*=active \lccode`*=`^ \catcode`\'=12 \catcode`"=active \lccode`"=`\'
\lowercase{\gdef\bbl@pr@m@s{\bbl@if@primes'\pr@@@s{\bbl@if@primes*^\pr@@@t\egroup}}}
\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).

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

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

\expandafter\def\csname OT1dqpos\endcsname{127}
\expandafter\def\csname T1dqpos\endcsname{4}
When the macro \f@encoding is undefined (as it is in plain \TeX) we define it here to expand to OT1

\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.

\begin{verbatim}
1571 \ifx\bbl@known@attribs\undefined
1572 \in@false
1573 \else
1574 \bbl@xin{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
1575 \fi
1576 \ifin@
1577 \bbl@warning{%
1578 You have more than once selected the attribute ‘##1’\%
1579 for language #1. Reported}%
1580 \else
1581 When we end up here the attribute is not selected before. So, we add it to the list of selected attributes and execute the associated \TeX-code.
1582 \bbl@exp{%
1583 \bbl@add@list\bbl@known@attribs{\bbl@tempc-##1}}%
1584 \edef\bbl@tempa{\bbl@tempc-##1}%
1585 \expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes%
1586 {\csname\bbl@tempc @attr@##1\endcsname}%
1587 {\@attrerr{\bbl@tempc}{##1}}%
1588 \fi}}
1589 \@onlypreamble\languageattribute
\end{verbatim}

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

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

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

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

\bbl@ifattributeset This internal macro has 4 arguments. It can be used to interpret \TeX-code based on whether a certain attribute was set. This command should appear inside the argument to \texttt{\AtBeginDocument} because the attributes are set in the document preamble, after babel is loaded.

The first argument is the language, the second argument the attribute being checked, and the third and fourth arguments are the true and false clauses.

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

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

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

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

```
\def\bbl@clear@ttribs{\ifx\bbl@attributes\@undefined\else\bbl@loopx\bbl@tempa\bbl@attributes\expandafter\bbl@clear@ttrib\bbl@tempa.\let\bbl@attributes\@undefined\fi}
```

```
\AtBeginDocument{\bbl@clear@ttribs}
```

### 4.7 Support for saving macro definitions

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

```
\def\babel@beginsave{\babel@savecnt\z@}
```

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

```
\newcount\babel@savecnt\babel@beginsave
```

```
\def\babel@save#1{\def\bbl@tempa{,#1,}% Clumsy, for Plain\expandafter\bbl@add\bbl@tempa\expandafter{\expandafter{,#1,}}\expandafter\in@\bbl@tempa\ifin@\bbl@add\bbl@savedextras{,#1,}\bbl@carg\let{\texttt{babel@}\number\babel@savecnt}#1\relax\toks@\expandafter{\originalTeX\let#1=}\bbl@exp{\def\originalTeX{\the\toks@\texttt{\the\toks@<\texttt{babel@}\number\babel@savecnt}\relax}}\advance\babel@savecnt\@ne^{4}\texttt{\originalTeX} has to be expandable, i.e. you shouldn't let it to \relax.
```

40
Some languages need to have \frenchspacing in effect. Others don't want that. The command \frenchspacing switches it on when it isn't already in effect and \nonfrenchspacing switches it off if necessary. A more refined way to switch the catcodes is done with ini files. Here an auxiliary macro is defined, but the main part is in \bbl@provide. This new method should be ideally the default one.

\def\bbl@frenchspacing{% 
\ifnum\the\sfcode`\relax=\@m
\let\bbl@nonfrenchspacing\relax
\else
\frenchspacing
\let\bbl@nonfrenchspacing\nonfrenchspacing
\fi}

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

4.8 Short tags

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

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

\bbl@trace{Hyphens}
\@onlypreamble\babelhyphenation
\AtEndOfPackage{%}
\newcommand\babelhyphenation[2][\@empty]{%
  \ifx\bbl@hyphenation@\relax
  \let\bbl@hyphenation@\@empty
  \fi
  \ifx\bbl@hyphlist\@empty\else
    \bbl@warning{You must not intermingle \string\selectlanguage\space and\%
      \string\babelhyphenation\space or some exceptions will not\%
      be taken into account. Reported}\%
  \fi
  \ifx\@empty#1%
    \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
  \else
    \bbl@vforeach{#1}{%
      \def\bbl@tempa{##1}%
      \bbl@fixname\bbl@tempa
      \bbl@iflanguage\bbl@tempa{%
        \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%\%
          \bbl@ifunset{bbl@hyphenation@\bbl@tempa}{}%\%
          \csname bbl@hyphenation@\bbl@tempa\@empty\endcsname\space}%
          (#2)}}%
  \fi}
\bbl@allowhyphens
This macro makes hyphenation possible. Basically its definition is nothing more than \nobreak
\hskip 0pt plus 0pt.\textsuperscript{3}
\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}
\babelhyphen
Macros to insert common hyphens. Note the space before @ in \babelhyphen. Instead of protecting it with \DeclareRobustCommand, which could insert a \relax, we use the same procedure as shorthands, with \active@prefix.
\newcommand\babelnullhyphen{\char\hyphenchar\font}
\def\babelhyphen{\active@prefix\babelnullhyphen\bbl@hyphen}
\def\bbl@hyphen{\@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
\def\bbl@hyphen@i#1#2{%
  \bbl@if_unset{bbl@hyphen@i@empty}\%
  \csname bbl@hyphen@i\@empty\endcsname{\discretionary{#2}}{#2}}%
\def\bbl@hyphen@i#1#2{%
  \csname bbl@hyphen@i@empty\endcsname}

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

\textsuperscript{3}TEX begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.
There should not be a discretionary after a hyphen at the beginning of a word, so it is prevented if preceded by a skip. Unfortunately, this does handle cases like “(-suffix)”. \nobreak is always preceded by \leavevmode, in case the shorthand starts a paragraph.

\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}

The following macro inserts the hyphen char.
\def\bbl@hyphenchar{\ifnum\hyphenchar\font=\m@ne\babelnullhyphen\else\char\hyphenchar\font\fi}

Finally, we define the hyphen “types”. Their names will not change, so you may use them in lfd’s.
After a space, the \mbox in \bbl@hy@nobreak is redundant.
\def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
\def\bbl@hy@@soft{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
\def\bbl@hy@hard{\bbl@usehyphen{\bbl@hyphenchar}}
\def\bbl@hy@@hard{\bbl@@usehyphen{\bbl@hyphenchar}}
\def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
\def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
\def\bbl@hy@repeat{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
\def\bbl@hy@@repeat{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
\def\bbl@hy@empty{\hskip\z@skip}
\def\bbl@hy@@empty{\discretionary{}{}{}}
\bbl@disc For some languages the macro \bbl@disc is used to ease the insertion of discretionaries for letters that behave ‘abnormally’ at a breakpoint.
\def\bbl@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bbl@allowhyphens}

4.10 Multiencoding strings

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

Tools But first, a tool. It makes global a local variable. This is not the best solution, but it works.
\def\bbl@trace{Multiencoding strings}
\def\bbl@toglobal#1{\global{\let#1#1}}

The following option is currently no-op. It was meant for the deprecated \SetCase.
\DeclareOption{nocase}{\let\bbl@opt@strings\@nnil % accept strings=value
\let\bbl@opt@strings\relax}
\DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
\DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
\def\BabelStringsDefault{generic}

\bbl@disc For some languages the macro \bbl@disc is used to ease the insertion of discretionaries for letters that behave ‘abnormally’ at a breakpoint.
\def\bbl@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bbl@allowhyphens}
Main command  This is the main command. With the first use it is redefined to omit the basic setup in subsequent blocks. We make sure strings contain actual letters in the range 128-255, not active characters.

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

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

1815 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1816 \let\SetString\@gobbletwo
1817 \let\bbl@stringdef\@gobbletwo
1818 \let\AfterBabelCommands\@gobble
1819 \ifx\@empty#1%
1820 \def\bbl@sc@label{generic}%
1821 \def\bbl@encstring##1##2{%
1822 \ProvideTextCommandDefault##1{##2}%
1823 \bbl@foglobal##1%
1824 \expandafter\bbl@foglobal\csname string?\string##1\endcsname%
There are two versions of \bbl@scswitch. The first version is used when ldfs are read, and it makes sure \verb!(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 \verb!\bbl@forlang! loops \verb!\bbl@L! but its body is executed only if the value is in \verb!\BabelLanguages! (inside babel) or \verb!\date!(language) is defined (after babel has been loaded). There are also two versions of \verb!\bbl@forlang!. The first one skips the current iteration if the language is not in \verb!\BabelLanguages!, and the second one skips undefined languages (after babel has been loaded).

\begin{verbatim}
\def\bbl@forlang#1#2{% 
  \bbl@for#1\bbl@L{% 
    \bbl@xin@{,#1,}{,\BabelLanguages,}{} % #1
\def\bbl@scswitch{%
  \let\bbl@sctest\in@true
  \else
    \let\bbl@sctest\in@false
\end{verbatim}
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{%
  \bbl@forlang\bbl@tempa{%  \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%  \bbl@ifunset{\bbl@LC}{\bbl@exp{%  \global\\bbl@add<br\bbl@G\bbl@tempa>{\bbl@scset\#1<br\bbl@LC>}}}%{}}\bbl@usehooks{stringprocess}{%\expandafter\bbl@stringdef\csname\bbl@LC\endcsname\expandafter{\BabelString}}}
```

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

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

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

```
⟨⟨∗ Macros local to BabelCommands ⟩⟩ ≡
\def\SetStringLoop##1##2{%
  \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
  \count@\z@
  \bbl@loop\bbl@tempa{##2}{{}}\count@\the\count@\relax}}}%
```

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

```
\def\bbl@aftercmds#1{%
  \toks@\expandafter\{\bbl@scafter#1\}%
\}
```

46
Case mapping  The command `\SetCase` is deprecated. Currently it consists in a definition with a hack just for backward compatibility in the macro mapping.

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

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

The following package options control the behavior of hyphenation mapping.
Initial setup to provide a default behavior if hyphenmap is not set.

\AtEndOfPackage{%\ifx\bbl@opt@hyphenmap\undefined
  \bbl@xin{},\{\bbl@language@opts\}%
\chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi}

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

\newcommand\setlocalecaption{% TODO. Catch typos.
  \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
\def\bbl@setcaption@x#1#2#3{% language caption-name string
  \bbl@trim@def\bbl@tempa{#2}%
  \bbl@xin{.template}{\bbl@tempa}%
  \ifin@
    \bbl@ini@captions@template{#3}{#1}%
  \else
    \edef\bbl@tempd{\expandafter\expandafter\expandafter\
      \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
    \bbl@xin@
    \{\expandafter\string\csname #2name\endcsname}%
    \bbl@tempd%
    \ifin% Renew caption
      \bbl@xin\{\string\bbl@scset\{\bbl@tempd}\%
    \else
      \edef\bbl@tempa{#2}%
      \bbl@exp%
      \\\bbl@ifsamestring\bbl@tempa\{\languagename}%
      \\\bbl@scset\<$\#1\#2name>$%
      \}%
  \else % Old way converts to new way
    \bbl@ifunset{#1#2name}%
    \bbl@funs\{#1#2name}%
    \bbl@exp%
    \\\bbl@add\{\captions#1\{\#2name\}<\#1\#2name>\}%
    \\\bbl@ifsamestring\bbl@tempa\{\languagename}%
    \\\bbl@add\{\#2name\{\<\#1\#2name>\}%
    \}}%
  \}%
\else % New way
  \bbl@xin\{\string\bbl@scset\{\bbl@tempd}\%
  \ifin% New way
    \bbl@exp%
    \\\bbl@add\{\captions#1\{\bbl@scset\<\#1\#2name>\}%
    \\\bbl@ifsamestring\bbl@tempa\{\languagename}%
    \\\bbl@scset\<\#1\#2name>\}%
    \}%
  \else % Old way, but defined in the new way
    \bbl@exp%
    \\\bbl@add\{\captions#1\{\<\#1\#2name>\}%
    \\\bbl@ifsamestring\bbl@tempa\{\languagename}%
    \\\bbl@add\{\<\#1\#2name>\}%
    \}%
  \}%
\fi%
\fi
% @namedef{#1#2name}(*3)%
\toks@expandafter\bbl@captionslist%
\bbl@exp\{\lq\#2name\}\\the\toks@%
\ifin\else
  \bbl@exp\\\bbl@add\\\bbl@captionslist\\<\#2name>\}%
\fi
\bbl@to@global\bbl@captionslist
4.11 Macros common to a number of languages

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

\set@low@box
\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.def.

4.12.1 Quotation marks

\quotedblbase
\quotesinglbase
\guillemetleft
\guillemetright

The guillemet characters are not available in OT1 encoding. They are faked. (Wrong names with o preserved for compatibility.)
Make sure that when an encoding other than OT1 or T1 is used these glyphs can still be typeset.

\ProvideTextCommandDefault{\guilsinglleft}{\UseTextSymbol{OT1}{\guilsinglleft}}
\ProvideTextCommandDefault{\guilsinglright}{\UseTextSymbol{OT1}{\guilsinglright}}

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

4.12.2 Letters

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.

\DeclareTextCommand{\ij}{OT1}{i\kern-0.02em\bbl@allowhyphens j}
\DeclareTextCommand{\IJ}{OT1}{I\kern-0.02em\bbl@allowhyphens J}
\DeclareTextCommand{\ij}{T1}{\char188}
\DeclareTextCommand{\IJ}{T1}{\char156}

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

\ProvideTextCommandDefault{\ij}{\UseTextSymbol{OT1}{\ij}}
\ProvideTextCommandDefault{\IJ}{\UseTextSymbol{OT1}{\IJ}}

4.12.2 Letters
The Croatian language needs the letters \dj and \DJ; they are available in the T1 encoding, but not in the OT1 encoding by default. Some code to construct these glyphs for the OT1 encoding was made available to me by Stipčević Mario, (stipcevic@olimp.irb.hr).

\begin{verbatim}
def\crrtic@{\hrule height0.1ex width0.3em}\def\crttic@{\hrule height0.1ex width0.33em}\def\ddj@{% \setbox0\hbox{d}\dimen@=\ht0\advance\dimen@1ex\dimen@.45\dimen@ \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@ \advance\dimen@ii.5ex\leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}}\def\DDJ@{% \setbox0\hbox{D}\dimen@=.55\ht0\dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@ \advance\dimen@ii.15ex % correction for the dash position\advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font\dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@ \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}}\providecommand{\dj}{\ddj@ d}\providecommand{\DJ}{\DDJ@ D}\makeatletter\def\providecommanddefault#1#2{\@ifstar{#2}{\@ifnextchar{}{#2}\@ifnextchar{\text@math}{#2}\@ifnextchar{\texttt}{#2}}}\makeatother
\end{verbatim}

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

\providecommanddefault{\dj}{\usefont{T1}{qm}{m}{n}d}{\dj}\providecommanddefault{\DJ}{\usefont{T1}{qm}{m}{n}D}{\DJ}

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

\providecommand{\SS}{\usefont{T1}{qm}{m}{n}SS}\providecommanddefault{\SS}{\usefont{T1}{qm}{m}{n}SS}

4.12.3 Shorthands for quotation marks

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

\glq\grq The ‘german’ single quotes.
\glqq\grqq The ‘german’ double quotes.
The 'french' single guillemets.
The 'french' double guillemets.

4.12.4 Umlauts and tremas

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

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

```
\umlauthigh
\umlautlow
```

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

```
\lower@umlaut
```

The following code fools T\TeX{}'s make\textunderscore accent procedure about the current x-height of the font to force another placement of the umlaut character. First we have to save the current x-height of the font, because we'll change this font dimension and this is always done globally. Then we compute the new x-height in such a way that the umlaut character is lowered to the base character. The value of .45ex depends on the METAFONT parameters with which the fonts were built. (Just try out, which value will look best.) If the new x-height is too low, it is not changed. Finally we call the \texttt{accent} primitive, reset the old x-height and insert the base character in the argument.

```
\lower@umlaut
```

\umlauthigh
\umlautlow
\lower@umlaut

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

```
\lower@umlaut
```

The following code fools T\TeX{}'s make\textunderscore accent procedure about the current x-height of the font to force another placement of the umlaut character. First we have to save the current x-height of the font, because we'll change this font dimension and this is always done globally. Then we compute the new x-height in such a way that the umlaut character is lowered to the base character. The value of .45ex depends on the METAFONT parameters with which the fonts were built. (Just try out, which value will look best.) If the new x-height is too low, it is not changed. Finally we call the \texttt{accent} primitive, reset the old x-height and insert the base character in the argument.

```
\lower@umlaut
```

\umlauthigh
\umlautlow
\lower@umlaut

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

```
\lower@umlaut
```

The following code fools T\TeX{}'s make\textunderscore accent procedure about the current x-height of the font to force another placement of the umlaut character. First we have to save the current x-height of the font, because we'll change this font dimension and this is always done globally. Then we compute the new x-height in such a way that the umlaut character is lowered to the base character. The value of .45ex depends on the METAFONT parameters with which the fonts were built. (Just try out, which value will look best.) If the new x-height is too low, it is not changed. Finally we call the \texttt{accent} primitive, reset the old x-height and insert the base character in the argument.

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

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

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

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

4.13 Layout

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

\bbl@trace{Bidi layout}
\providecommand\IfBabelLayout[3]{#3}%
\newcommand\BabelPatchSection[1]{%
  \@ifundefined{#1}{}{%
    \bbl@exp{\let<\bbl@ss@#1><#1>}%
    \namedef{#1}{%
    \@ifstar{\bbl@presec@s{#1}}{\@dblarg{\bbl@presec@x{#1}}}}}
\def\bbl@presec@x#1[#2]#3{%
  \bbl@exp{%
  \select@language@x{\bbl@main@language}%
  \bbl@cs{sspre@#1}%
  \bbl@cs{ss@#1}%
  {\foreignlanguage{\languagename}{\unexpanded{#2}}}%
  {\foreignlanguage{\languagename}{\unexpanded{#3}}}%
  \select@language@x{\languagename}}}
\def\bbl@presec@s#1[#2]#3{%
  \bbl@exp{%
  \select@language@x{\bbl@main@language}%
  \bbl@cs{sspre@#1}%
  \bbl@cs{ss@#1}%
  {\foreignlanguage{\languagename}{\unexpanded{#2}}}%
  {\foreignlanguage{\languagename}{\unexpanded{#3}}}%
  \select@language@x{\languagename}}}
\IfBabelLayout{sectioning}{%
  \BabelPatchSection{part}%
  \BabelPatchSection{chapter}%
  \BabelPatchSection{section}%
  \BabelPatchSection{subsection}%
  \BabelPatchSection{subsubsection}%
}
4.14 Load engine specific macros

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

4.15 Creating and modifying languages

Continue with \LaTeX only.
\texttt{\textbackslash babelprovide} is a general purpose tool for creating and modifying languages. It creates the language infrastructure, and loads, if requested, an ini file. It may be used in conjunction to previously loaded \texttt{ldf} files.
At this point all parameters are defined if `import`. Now we execute some code depending on them. But what about if nothing was imported? We just set the basic parameters, but still loading the whole ini file.

```latex
\bblload@basic[#2]\fi
\% == include == TODO
\% \ifx\bbllincluded@inis@empty\else
\bbllreplace\bbllincluded@inis{\}{}\%
\bbllforeach\bbllincluded@inis{%
\openin\bbllreadstream=babel-#1.ini
\bbllextend@ini[#2]%
\closein\bbllreadstream
\fi
\% Post tasks
\% ------------
\% == subsequent calls after the first provide for a locale ==
\% \ifx\bbllinidata@empty\else\bbllextend@ini[#2]\fi
\% == ensure captions ==
\% \ifx\bbllKVP@captions@empty\else\bbllifunset{bbl@extracaps@#2}{}\else\bbllexp{\bbllif\selectlanguage{#2}{}}\fi\fi
\% == script, language ==
\% \ifx\bbllKVP@script@empty\else
\bbllcsarg\edef{sname@#2}{\bbllKVP@script}\fi
\% \ifx\bbllKVP@language@empty\else
\bbllcsarg\edef{lname@#2}{\bbllKVP@language}\fi
\% \ifcase\bbllengine\or
\bbllifunset{bbl@chrng@language}{}\else
\directlua{Babel.set_chranges_b('sbcp', 'chrng')}\fi
\bbllifcase\bbllengine\or
\bbllifunset{bbl@chrng@language}{}\else
{\directlua{
Babel.set_chranges_b('sbcp', 'chrng')}}\fi
\bbllif\bbllchrng@language\else
\bbllchrng@language\fi
\bbllchrng@language\fi
\ifx\bbllKVP@onchar@empty\else
\bbllluahyphenate
\bbllexp{\AddToHook{env/document/before}{\selectlanguage{#2}}\directlua{
if Babel.locale_mapped == nil then
Babel.locale_props[\the\localeid].letters = false
```
\ifin@
  \directlua{
    Babel.locale_props[\the\localeid].letters = true
  }
\fi
\bbl@xin@{ letters }{ \bbl@KVP@onchar\space}

% \ifin@
  \bbl@exp{\\bbl@add\\bbl@starthyphens
{\bbl@patterns@lua{\languagename}}}%
% \ifin@
  \bbl@ifunset{bbl@lsys@languagename}{\bbl@provide@lsys{\languagename}}{}
  \bbl@ifunset{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}']
      Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@languagename}\space
    end
  }
% \ifin@
  \bbl@exp{\\bbl@add\\bbl@mapselect{\\bbl@mapdir{\languagename}}}{}
% \ifin@
  \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
  \bbl@ifunset{bbl@lsys@languagename}{\bbl@provide@lsys{\languagename}}{}
  \bbl@ifunset{bbl@wdir@languagename}{\bbl@provide@dirs{\languagename}}{}
  \ifx\bbl@mapselect\@undefined % TODO. See onchar.
    \AtBeginDocument{%}
      \bbl@patchfont{{\bbl@mapselect}}%
      ({\selectfont})%
    \def\bbl@mapselect{%}
      \let\bbl@mapselect\relax
      \edef\bbl@prefontid{\fontid\font}
    \def\bbl@mapdir##1{%}
      \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
    \fi
  % == mapfont ==
  % For bidi texts, to switch the font based on direction
  \ifx\bbl@KVP@mapfont\@nil\else
    \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
    {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\%
      mapfont. Use 'direction'.%}}
    (See the manual for details.))%
  \fi
  \bbl@ifunset{bbl@lsys@languagename}{\bbl@provide@lsys{\languagename}}{}
  \bbl@ifunset{bbl@wdir@languagename}{\bbl@provide@dirs{\languagename}}{}
  \ifx\bbl@mapselect\@undefined % TODO. See onchar.
    \AtBeginDocument{%}
  \fi
% \ifin@
%  \bbl@exp{\\bbl@add\\bbl@mapselect{\\bbl@mapdir{\languagename}}}{}
% \ifin@
%  % TODO - catch non-valid values
% \ifin@
%  % == mapfont ==
%  % For bidi texts, to switch the font based on direction
%  \ifx\bbl@KVP@mapfont\@nil\else
%    \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
%    {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\%
%      mapfont. Use 'direction'.%}}
%    (See the manual for details.))%
Depending on whether or not the language exists (based on \date<language>), we define two macros. Remember \bbl@startcommands opens a group.

\def\bbl@provide@new#1{\bbl@exp{\bbl@tempe\bbl@KVP@calendar\space\empty\@}}
\def\bbl@tempe##1.##2.##3\@{\def\bbl@tempc{##1}\def\bbl@tempb{##2}}
\expandafter\bbl@tempe\bbl@tempa..\@\bbl@csarg\edef{calpr@\languagename}{\ifx\bbl@tempc\@empty\else calendar=\bbl@tempc\fi\ifx\bbl@tempb\@empty\else ,variant=\bbl@tempb\fi}\%\def\bbl@provide@new{\bbl@provide@new}\%\def\bbl@provide@new{\bbl@provide@new}\%\def\bbl@provide@new{\bbl@provide@new}\%\def\bbl@provide@new{\bbl@provide@new}\%
Load the basic parameters (ids, typography, counters, and a few more), while captions and dates are left out. But it may happen some data has been loaded before automatically, so we first discard the saved values. (TODO. But preserving previous values would be useful.)
The hyphenrules option is handled with an auxiliary macro. This macro is called in three cases:
when a language is first declared with babelprovide, with hyphenrules and with import.

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

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

\def\bbl@loop@ini{\loop\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
    \readlinechar\m@ne
    \read\bbl@readstream to \bbl@line
    \readlinechar`^^M
    \ifx\bbl@line\empty\else
        \expandafter\bbl@line\linebreak\bbl@iniline\bbl@line\bbl@iniline
    \fi
\repeat}
(There is no ini file for the requested language). Perhaps you misspelled it or your installation is not complete.

Fix the name or reinstall babel.

\else

% == Store ini data in \bbl@inidata ==
\catcode`\[=12 \catcode`\]=12 \catcode`\%=12 \catcode`\-=12
\catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
\bbl@info{Importing \texttt{\#2}\texttt{\textendash}font and identification \textbf{or basic \textbackslash fi data for }\texttt{\languagename}\%
from babel-\texttt{\#1}. Reported}\%
\ifnum\#2=\z@
\global\let\bbl@inidata\@empty
\let\bbl@inistore\bbl@inistore@min % Remember it's local
\fi
\def\bbl@section{identification}%
\bbl@exp{\\bbl@inistore tag.ini=\#1\\@@}
\bbl@inistore load.level=\#2\\@@
\bbl@loop@ini
% == Process stored data ==
\bbl@csarg\xdef{lini@\languagename}{\#1}\
\bbl@read@ini@aux
% == 'Export' data ==
\bbl@ini@exports{\#2}\
\global\bbl@csarg\let{inidata@\languagename}\bbl@inidata\
\global\bbl@inidata@empty\
\bbl@exp{\\\bbl@add@list\\\bbl@ini@loaded{\languagename}}\
\global\bbl@inidata\bbl@ini@loaded
\fi
\closein\bbl@readstream
\def\bbl@read@ini@aux{%
\let\bbl@savestrings\@empty
\let\bbl@savetoday\@empty
\let\bbl@savedate\@empty
\def\bbl@elt##1##2##3{%\def\bbl@section{##1}%
\in@=date.}{=##1}% Find a better place
\ifin@
\bbl@ifunset{bbl@inikv@##1}%
{\bbl@ini@calendar{##1}}%
\bbl@ifunset{bbl@inikv@##1}{}%
\csname bbl@inikv@##1\endcsname{##2}{##3}}%
\bbl@inidata}

A variant to be used when the ini file has been already loaded, because it's not the first \babelprovide for this language.
\def\bbl@extend@ini@aux#1{%
\bbl@startcommands*{#1}{captions}%
% Activate captions/... and modify exports
\bbl@csarg\def{inikv@captions.licr}{#1}\#2{}\
\setlocalecaption{#1}{#1}{#2}%
\bbl@inikv@captions#1\#2{}%}
\def\bbl@inikv@captions#1\#2{%\setlocalecaption{#1}{#1}{#2}%
\def\bbl@stringdef{#1\#2}{#3}%
\bbl@exportkey{#1\#2}{#3}%
\bbl@funset{bbl@kv\#2}{}
{\expandafter\ifx\csname bbl@inikv@#1\endcsname@empty\else
\bbl@exp{\global\let\bbl@#1@languagename=\\\bbl@kv\#2}{}\fi}%
% As with \bbl@read@ini, but with some changes
As a somewhat hackish tool to handle calendar sections. TODO. To be improved.

A key with a slash in \bbl@provide 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.
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.

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

\begin{verbatim}
% Identification always exported
\bbl@iniwarning{}%
\ifcase\bbl@engine
  \bbl@iniwarning{.pdflatex}%
  \or
  \bbl@iniwarning{.lualatex}%
  \or
  \bbl@iniwarning{.xelatex}%
  \fi%
\bbl@exportkey{llevel}{identification.load.level}{}%
\bbl@exportkey{elname}{identification.name.english}{}%
(\csname bbl@elname@languagename@endcsname)%
\bbl@exportkey{tbcp}{identification.tag.bcp47}{}%
% Somewhat hackish. TODO:
\bbl@exportkey{casing}{identification.tag.bcp47}{}%
\bbl@exportkey{lbcp}{identification.language.tag.bcp47}{}%
\bbl@exportkey{lotf}{identification.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{sotf}{identification.script.tag.opentype}{DFLT}%
\bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
\bbl@exportkey{vbcp}{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
\if\bbl@bcptoname
  \bbl@csarg\xdef{bcp@map@\bbl@cl{tbcp}}{\languagename}\
\fi
\ifnum#1>\z@
% 0 = only info, 1, 2 = basic, (re)new
  \bbl@exportkey{calpr}{date.calendar.preferred}{}%
  \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
  \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
  \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
  \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
  \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
  \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
  \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
  \bbl@exportkey{intsp}{typography.intraspace}{}%
  \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
  \bbl@exportkey{chrng}{characters.ranges}{}%
\fi
\end{verbatim}
A shared handler for key=val lines to be stored in `\bbl@kv@<section>.<key>`.

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

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

```latex
\let\bbl@inikv@identification\bbl@inikv
\let\bbl@inikv@date\bbl@inikv
\let\bbl@inikv@typography\bbl@inikv
\let\bbl@inikv@numbers\bbl@inikv
```

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

```latex
\def\bbl@maybextx{-\bbl@csarg\ifx{extx@\languagename}\@empty x-\fi}
\def\bbl@inikv@characters#1#2{%
  \bbl@ifsamestring{#1}{casing}% eg, casing = uV
  {\bbl@exp{\g@addto@macro\bbl@release@casing{\bbl@casemapping{}{\languagename}{\unexpanded{#2}}}}}%
  {\in@{$casing.}{$#1}% eg, casing.Uv = uV
    \ifin@
      \lowercase{\def\bbl@tempb{#1}}%
      \bbl@replace\bbl@tempb{casing.}{}%
      \bbl@exp{\g@addto@macro\bbl@release@casing{\bbl@casemapping{\bbl@maybextx\bbl@tempb}{\languagename}{\unexpanded{#2}}}}}%
    \else
      \bbl@inikv{#1}{#2}%
    \fi
  }
}
```

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

```latex
\def\bbl@inikv@counters#1#2{%
  \bbl@ifsamestring{#1}{digits}%
  {\bbl@error{The counter name 'digits' is reserved for mapping}\
    \decimal digits}%
  {Use another name.}\
}
```

```latex
\def\bbl@trim@def#1*#2\@empty#3\@empty{\in@{.1$}{#1$}\
  \ifin@
    \bbl@replace\bbl@tempb{.1}{}\
    \bbl@csarg\protected@xdef{cntr\bbl@tempb \languagename}{\bbl@alphnumeral{\bbl@tempb}}%
  \else
    \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
  \fi
}
```

```latex
\def\bbl@tempa{.1\@empty}\def\bbl@tempb{.1\@empty}\def\bbl@tempc{.1\@empty}
```

```latex
\def\bbl@tempd{.1\@empty}\def\bbl@tempe{.1\@empty}\def\bbl@tempf{.1\@empty}
```

```latex
\def\bbl@temps{.1\@empty}\def\bbl@tempg{.1\@empty}\def\bbl@temph{.1\@empty}
```

```latex
\def\bbl@tempi{.1\@empty}\def\bbl@tempj{.1\@empty}\def\bbl@tempk{.1\@empty}
```

```latex
\def\bbl@tempm{.1\@empty}\def\bbl@tempn{.1\@empty}\def\bbl@temps{.1\@empty}
```

```latex
\def\bbl@tempo{.1\@empty}\def\bbl@tempp{.1\@empty}\def\bbl@tempq{.1\@empty}
```

```latex
\def\bbl@tempr{.1\@empty}\def\bbl@temps{.1\@empty}\def\bbl@temps{.1\@empty}
```

```latex
\def\bbl@tempu{.1\@empty}\def\bbl@tempv{.1\@empty}\def\bbl@tempw{.1\@empty}
```

```latex
\def\bbl@tempx{.1\@empty}\def\bbl@tempy{.1\@empty}\def\bbl@tempz{.1\@empty}
```
Now captions and captions.licr, depending on the engine. And below also for dates. They rely on a few auxiliary macros. It is expected the ini file provides the complete set in Unicode and LICR, in that order.

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

Labels. Captions must contain just strings, no format at all, so there is new group in ini files.
To show correctly some captions in a few languages, we need to patch some internal macros, because the order is hardcoded. For example, in Japanese the chapter number is surrounded by two string, while in Hungarian is placed after. These replacement works in many classes, but not all. Actually, the following lines are somewhat tentative.
Date. Arguments (year, month, day) are not protected, on purpose. In  \today , arguments are always gregorian, and therefore always converted with other calendars. TODO. Document

\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}\bbl@replace\bbl@tempe{}{}\bbl@replace\bbl@tempe{convert=}{}\let\bbl@ld@calendar\@empty\let\bbl@ld@variant\@empty\let\bbl@ld@convert\relax\bbl@foreach\bbl@tempe{\bbl@tempb##1@@}\bbl@replace\bbl@ld@calendar{gregorian}{}\ifx\bbl@ld@calendar\@empty\else\ifx\bbl@ld@convert\relax\else\babelcalendar[\bbl@they-\bbl@them-\bbl@thed]{\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed\fi\fi\@nameuse{bbl@precalendar}% Remove, eg, +, -civil (-ca-islamic)\edef\bbl@calendar{% Used in \month..., too\bbl@ld@calendar}
Dates will require some macros for the basic formatting. They may be redefined by language, so “semi-public” names (camel case) are used. Oddly enough, the CLDR places particles like “de” inconsistently in either the date or in the month name. Note after \bbl@replace \toks@ contains the resulting string, which is used by \bbl@replace@finish@iii (this implicit behavior doesn’t seem a good idea, but it’s efficient).
range 0-9999.}%

\newcommand\BabelDateyyyy[1][{{\number#1}}] % TODO - add leading 0
\newcommand\BabelDateU[1][{{\number#1}}]
\def\bbl@replace@finish@iii#1{{\bbl@exp{\def\#1####1####2####3\the\toks@}}}
\def\bbl@TG@@date{{\bbl@replace\bbl@toreplace{[\ ]}{\BabelDateSpace{}}\bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}\bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}\bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}\bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}\bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}\bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}\bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}\bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}\bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}\bbl@replace\bbl@toreplace{[U]}{\BabelDateU{####1}}\bbl@replace\bbl@toreplace{[y|}{\bbl@datecntr[####1|}\bbl@replace\bbl@toreplace{[U|}{\bbl@datecntr[####1|}\bbl@replace\bbl@toreplace{[m|}{\bbl@datecntr[####2|}\bbl@replace\bbl@toreplace{[d|}{\bbl@datecntr[####3|}\bbl@replace@finish@iii\bbl@toreplace}
\def\bbl@datecntr\#1\#2\#3\#4\#5\relax{\#1\[#2\]{\#3}{\#4}{\#5}}
\begingroup % A hack. TODO. Don’t require an specific order
\catcode`%=12
\catcode`&=14
\gdef\bbl@transforms#1#2#3{&}\directlua{
local str = 
str = str:gsub('%.%d+%.%d+$', '')
token.set_macro('babeltempa', str)
}\def\babeltempc{}
\def\bbl@transforms#1#2#3{&}\def\bbl@transforms#1#2#3{&\directlua{
local t = {}
for m in string.gmatch('##1..', '(.-):') do
  table.insert(t, m)
end
table.remove(t)
token.set_macro('babeltempc', ',fonts=' .. table.concat(t, ' '))\directlua{\font:font:transform syntax
\local t = {}
for m in string.gmatch('#1..', '(.-):') do
  table.insert(t, m)
end
table.remove(t)
token.set_macro('babeltempc', ',fonts=' .. table.concat(t, ' '))\directlua{\attribute syntax
local str = string.match([[\bbl@KVP@transforms]],}
if str == nil then
    token.set_macro('babeltempb', '')
else
    token.set_macro('babeltempb', ',attribute=' .. str)
end
}
\toks@{#3}
\bbl@exp{\
\g@addto@macro\bl@release@transforms{\
\relax &% Closes previous \bbl@transforms@aux
\bbl@transforms@aux
\#1{label=\babeltempla\babeltempb\babeltempc}&%
\language}{\the\toks@}}&%
\else
\g@addto@macro\bbl@release@transforms{, {#3}}&%
\fi
\fi}
\endgroup

Language and Script values to be used when defining a font or setting the direction are set with the following macros.
\def\bbl@provide@lsys#1{%
\bbl@ifunset{bbl@lname@#1}{\bbl@load@info{#1}}{}% \bbl@csarg\let{lsys@#1}@empty
\bbl@ifunset{bbl@lname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}% \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}% \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}% \bbl@ifunset{bbl@lname@#1}{}% \bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}% \ifcase\bbl@engine\or\or
\bbl@ifunset{bbl@prehc@#1}{}% \if\bbl@cs{prehc@#1}\@undefined\global\let\bbl@xenohyph\bbl@xenohyph@d
\AtBeginDocument{\bbl@patchfont{\bbl@xenohyph}{\expandafter\select@language\expandafter{\language}}}}% \fi
\bbl@exp{\bbl@xenohyph@d}
\bbl@ifset{bbl@prehc@\language}{\ifnum\hyphenchar\font=\defaulthyphenchar\iffontchar\font\bbl@cl{prehc}\relax
\hyphenchar\font\bbl@cl{prehc}\relax
\else\iffontchar\font=200B\hyphenchar\font=200B\else\else
\bbl@warning{Neither 0 nor ZERO WIDTH SPACE are available\%
in the current font, and therefore the hyphen\%
will be printed. Try changing the fontspec's\%
'HyphenChar' to another value, but be aware\%
this setting is not safe (see the manual).\%
Reported)\%}
\hyphenchar\font\defaulthyphenchar
The following ini reader ignores everything but the identification section. It is called when a font is defined (i.e., when the language is first selected) to know which script/language must be enabled. This means we must make sure a few characters are not active. The ini is not read directly, but with a proxy tex file named as the language (which means any code in it must be skipped, too).

A tool to define the macros for native digits from the list provided in the ini file. Somewhat convoluted because there are 10 digits, but only 9 arguments in \TeX. Non-digits characters are kept.

The first macro is the generic “localized” command.

Alphabetic counters must be converted from a space separated list to an \ifcase structure.
being the number of digits in the number to be converted. This explains the reverse set 76543210.
Digits above 10000 are not handled yet. When the key contains the subkey .F., the number after is
treated as an special form (see babel-he.ini, for example).

\newcommand{localenumeral}[2]{{\bbl@cs{cntr@#1\@\languagename}{#2}}}\def{\bbl@localecntr#1{\bbl@localenumeral{#1}{1}}}\newcommand{localecounter}[2]{%\expandafter{\bbl@localecntr}{\number{\csname c@#2\endcsname}{#1}}}\def{\bbl@alphnumeral#1#2}{\expandafter{\bbl@alphnumeral@i}\number#2\@@{#1}}\def{\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9}{\ifcase#9\or % Currently <10000, but prepared for bigger\bbl@alphnumeral@ii{#9}000000#1\or\bbl@alphnumeral@ii{#9}00000#1#2\or\bbl@alphnumeral@ii{#9}0000#1#2#3\or\bbl@alphnumeral@ii{#9}000#1#2#3#4\else\bbl@alphnumeral@invalid{>9999}\fi}%\def{\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8}{%\bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8\@@{#1}}{%\bbl@cs{cntr@#1.4\@\languagename}{#5}\bbl@cs{cntr@#1.3\@\languagename}{#6}\bbl@cs{cntr@#1.2\@\languagename}{#7}\bbl@cs{cntr@#1.1\@\languagename}{#8}\ifnum#6#7#8>-z\@ % TODO. An ad hoc rule for Greek. Ugly.\bbl@ifunset{bbl@cntr@#1.S.321\@\languagename}{\bbl@cs{cntr@#1.S.321\@\languagename}}\fi}{\bbl@cs{cntr@#1.F.\number#5#6#7#8\@@{#1}}}%\def{\bbl@alphnum@invalid#1}{%\bbl@error{Alphabetic numeral too large (#1)}%\ifnum#1>-z\@ % Currently this is the limit.\}%The information in the identification section can be useful, so the following macro just exposes it
with a user command.\def{\bbl@localeinfo#1#2}{%\bbl@ifunset{bbl@info@#2}{#1}%\bbl@ifunset{bbl@cs{\csname bbl@info@#2\endcsname\@\languagename}}{#1}}\newcommand{localeinfo}[1]{%\ifx*#1\@empty % TODO. A bit hackish to make it expandable.\else\bbl@localeinfo\{#1\}%\bbl@error{I've found no info for the current locale.\nThe corresponding ini file has not been loaded\Perhaps it doesn't exist}%\bbl@afterelse{\bbl@localeinfo\{#1\}}%\fi}\def{\bbl@localeinfo#1}{%\bbl@ifunset{bbl@info@name.locale}{lcname}\bbl@ifunset{bbl@info@tag.ini}{lini}\bbl@ifunset{bbl@info@name.english}{elname}\bbl@ifunset{bbl@info@name.opentype}{lname}\bbl@ifunset{bbl@info@tag.bcp47}{tbcp}\bbl@ifunset{bbl@info@language.tag.bcp47}{lbcp}\bbl@ifunset{bbl@info@script.name}{esname}\bbl@ifunset{bbl@info@script.name.opentype}{sname}\bbl@ifunset{bbl@info@script.tag.bcp47}{sbcp}\bbl@ifunset{bbl@info@script.tag.opentype}{sotf}\bbl@ifunset{bbl@info@region.tag.bcp47}{rbcp}\bbl@ifunset{bbl@info@variant.tag.bcp47}{vbcp}
\LaTeX needs to know the BCP 47 codes for some features. For that, it expects \textsc{BCPdata} to be defined. While language, region, script, and variant are recognized, extension. \langle type \rangle for singletons may change.

The parser for \texttt{casing} and \texttt{casing.\langle variant \rangle}.

The parser for casing and casing. \langle variant \rangle.

\texttt{\ifcase\bbl@engine % Converts utf8 to its code (expandable)}
\texttt{\def\bbl@utftocode#1{\the\numexpr\decode@UTFviii#1\relax}}
\texttt{\else}
\texttt{\def\bbl@utftocode#1{\expandafter\string#1}}
\texttt{\fi}
\texttt{\ifx\renewcommand\@undefined\else % For plain. TODO. It’s a quick fix}
\texttt{\renewcommand\BCPdata[]{\bbl@bcpdata@i\texttt{#1@empty}}} 
\texttt{\def\bbl@bcpdata@i#1#2#3#4#5#6\texttt{\@empty}{\@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}{\bbl@bcpdata@ii{#6}\texttt{\bbl@main@language}}}{\bbl@bcpdata@ii{#1#2#3#4#5#6}\texttt{\languagename}}}{\bbl@bcpdata@i#1\texttt{\@empty}}} 
\texttt{\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} 
\texttt{\edef\bbl@templ{\@nameuse{bbl@casing@#2}#1}% Language code}
\texttt{\def\bbl@tempe{0}% Mode (upper/lower...)
\def\bbl@tempc{#3 }% Casing list}
\expandafter\bbl@tempa\bbl@tempc\texttt{\@empty}}
\texttt{\def\bbl@casemapping@i#1%}
\texttt{\def\bbl@tempe{0}% Mode (upper/lower...)
\def\bbl@tempc{#3 }% Casing list}
% \edef\bbl@templ{\@nameuse{bbl@casing@#2}#1% Language code}
\texttt{\expandafter\bbl@casemapping@ii\bbl@tempc\texttt{\@empty}}
\texttt{\edef\bbl@templ{\@nameuse{bbl@casing@#2}#1% Language code}}
\texttt{\expandafter\bbl@casemapping@i#1%}
\texttt{\def\bbl@tempe{0}% Mode (upper/lower...)}
\texttt{\def\bbl@tempc{#3 }% Casing list}
\expandafter\bbl@casemapping@ii\bbl@tempc\texttt{\@empty}}
\texttt{\edef\bbl@templ{\@nameuse{bbl@casing@#2}#1% Language code}}
\texttt{\expandafter\bbl@casemapping@i#1%}
\texttt{\def\bbl@tempe{0}% Mode (upper/lower...)
\def\bbl@tempc{#3 }% Casing list}
\expandafter\bbl@casemapping@ii\bbl@tempc\texttt{\@empty}}
With version 3.75, \texttt{BabelEnsureInfo} is executed always, but there is an option to disable it.

```latex
\DeclareUppercaseMapping{\bbl@utftocode{#1}}{#2}
\DeclareLowercaseMapping{\bbl@utftocode{#1}}{#2}
\DeclareTitlecaseMapping{\bbl@utftocode{#1}}{#2}
\fi
\fi}
```

More package options:

```latex
\DeclareOption{ensureinfo=off}{}
```

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

```latex
\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}\bbl@getproperty@s{#1}{#2}{#3}\ifx#1\relax\bbl@error{Unknown key for locale '#2':\%#3\%\string#1 will be set to \relax}{Perhaps you misspelled it.}\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.

```latex
\newcommand\babeladjust[1]{% TODO. Error handling.
\def\bbl@error{\typeout{Unknown key for locale '##1':\%##3\%\string##1 will be set to \relax}{Perhaps you misspelled it.}}
\def\ShowLocaleProperties#1{\typeout{}\typeout{*** Properties for language '##1' ***}\def\bbl@elt##1##2##3{\typeout{##1/##2 = ##3}}\@nameuse{bbl@inidata@##1}}
```
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.

\newlabel 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.

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

\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.
\@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\@tempa{#3}\@safe@activesfalse\org@@citex[#1]{#2}}}{}\@ifpackageloaded{cite}{\def\@citex[#1]#2{\@safe@activestrue\org@@citex[#1]{#2}}}{}\nocite


\nocite

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

\AtBeginDocument{\@ifpackageloaded{cite}{\def\@citex[#1][#2]{\@safe@activestrue\org@nocite[#1][#2]}\@ifpackageloaded{cite}{\def\@citex[#1][#2][#3]{\@safe@activestrue\org@nocite[#1][#2][#3]}}}
\textbf{\texttt{\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 \texttt{@safe@activestru}e is in effect. This switch needs to be reset inside the \texttt{\box} which contains the citation label. In order to determine during .aux file processing which definition of \texttt{\bibcite} is needed we define \texttt{\bibcite} in such a way that it redefines itself with the proper definition. We call \texttt{\bbl@cite@choice} to select the proper definition for \texttt{\bibcite}. This new definition is then activated.

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

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

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

\textbf{\texttt{\markright}} Because the output routine is asynchronous, we must pass the current language attribute to the head lines. To achieve this we need to adapt the definition of \texttt{\markright} and \texttt{\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.

\textbf{\texttt{\markright}} Because the output routine is asynchronous, we must pass the current language attribute to the head lines. To achieve this we need to adapt the definition of \texttt{\markright} and \texttt{\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.

(Asof Oct 2019, \LaTeX stores the definition in an intermediate macro, so it's not necessary anymore, but it's preserved for older versions.)

\ifx\@mkboth\markboth
\def\bbl@tempc{\let\@mkboth\markboth}
\else
\def\bbl@tempc{\let\@mkboth\markboth}
\fi % end ifbbl@single, end \IfBabelLayout

\section{Preventing clashes with other packages}

\subsection{ifthen}

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

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

\begin{verbatim}
\ifthenelse{\isodd{\pageref{some:label}}}{}{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 \isodd 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.
5.3.2 varioref

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

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

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.

5.3.3 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.

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

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

\let\asciiencoding\@empty
\AtBeginDocument{\def\@elt#1{,#1,}\
\edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}\let\@elt\relax\let\bbl@tempb\@empty\def\bbl@tempc{OT1}\bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
\bbl@ifunset{T@#1}{}{\def\bbl@tempb{#1}}}\bbl@foreach\bbl@tempa{\bbl@xin@{,#1,}{,\BabelNonASCII,}\ifin@\def\bbl@tempb{#1}% Store last non-ascii\else\bbl@xin@{,#1,}{,\BabelNonText,}\ifin@\else\def\bbl@tempc{#1}% Store last ascii\fi\fi}\ifx\bbl@tempb\@empty\bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,}\fi\ifin@\def\bbl@tempb{#1}% Store last ascii\fi\fi\let\asciiencoding\bbl@tempc\renewcommand\ensureascii[1]{\fontencoding{\asciiencoding}\selectfont#1}\DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}\DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}
Now comes the old deprecated stuff (with a little change in 3.9l, for fontspec). The first thing we need to do is to determine, at \begin{document}, which latin fontcoding to use.

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

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

But this might be overruled with a later loading of the package fontenc. Therefore 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}}%\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.

\DeclareTextFontCommand{\textlatin}{%\let\textlatin@\leavevmode{\latintext #1}}

\DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}

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

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

5.5 Basic bidi support

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

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

- \texttt{pdftex} provides a minimal support for bidi text, and it must be done by hand. Vertical typesetting is not possible.
- \texttt{xetex} is somewhat better, thanks to its font engine (even if not always reliable) and a few additional tools. However, very little is done at the paragraph level. Another challenging problem is text direction does not honour \texttt{TeX} grouping.
- \texttt{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 \texttt{Lua\TeX-ja} shows, vertical typesetting is possible, too.

\begin{verbatim}
\bbl@trace{Loading basic (internal) bidi support}
\ifodd\bbl@engine
\else % TODO. Move to \texttt{txtbabel}
\ifnum\bbl@bidimode=100 \ifnum\bbl@bidimode<200 % Any xe+lua bidi=
\bbl@error
{The bidi method 'basic' is available only in\%
luatex. I'll continue with 'bidi=default', so\%
expect wrong results}%
{See the manual for further details.}%
\let\bbl@beforeforeign\leavevmode
\AtEndOfPackage{%
\EnableBabelHook{babel-bidi}%
\bbl@xebidipar}
\def\bbl@loadxebidi#1{%
\ifx\RTLfootnotetext\@undefined
\AtEndOfPackage{%
\EnableBabelHook{babel-bidi}%
\bbl@loadfontspec % bidi needs fontspec%
\usepackage#1{bidi}%
\let\bbl@digitsdotdash\DigitsDotDashInterCharToks
\def\DigitsDotDashInterCharToks{% See the 'bidi' package
\ifnum\@nameuse{bbl@wdir@\languagename}=	w@ % 'AL' bidi
\bbl@digitsdotdash % So ignore in 'R' bidi
\fi}}%
\fi}
\ifnum\bbl@bidimode>200 % Any xe bidi=
\ifcase\expandafter\gobbleto\the\bbl@bidimode\or
\bbl@tentative{bidi=bidi}%
\bbl@loadxebidi()%
\or
\bbl@loadxebidi[[rldocument]]
\or
\bbl@loadxebidi()
\fi
\fi
\fi\fi
\ifnum\bbl@bidimode=200 % Any xe bidi=
\ifcase\expandafter\gobbleto\the\bbl@bidimode\or
\bbl@tentative{bidi=bidi}%
\bbl@loadxebidi()%
\or
\bbl@loadxebidi([rldocument])
\or
\bbl@loadxebidi()
\fi
\fi
\fi
\fi
\fi
\fi
\ifnum\bbl@bidimode=\@ne % Any bidi= except default=1
\let\bbl@beforeforeign\leavevmode
\ifodd\bbl@engine
\newattribute\bbl@attr@dir
\directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
\bbl@exp{\output{\bodydir\pagedir\the\output}}
\fi
\AtEndOfPackage{%
\EnableBabelHook{babel-bidi}%
\ifodd\bbl@engine\else
\bbl@xebidipar
\fi}
\end{verbatim}
Now come the macros used to set the direction when a language is switched. First the (mostly) common macros.

\def\bbl@alscripts{%Arabic,Syriac,Thaana,}
\def\bbl@rscripts{%Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
Old Hungarian,Lydia,Mandaean,Manichaean,%
Meroitic Cursive,Meroitic,Old North Arabian,%
Nabataean,N’Ko,Orkhon,Palmrenee,Inscriptional Pahlavi,%
Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
Old South Arabian,}%
\def\bbl@provide@dirs#1{%\bbl@xin{{\csname bbl@sname@#1\endcsname}}{{\bbl@alscripts}{\bbl@rscripts}}%\ifin@
  \global\bbl@csarg\chardef{wdir@#1}\@ne
  \bbl@xin{{\csname bbl@sname@#1\endcsname}}{{\bbl@alscripts}}%\ifin@
  \global\bbl@csarg\chardef{wdir@#1}\tw@
  \else
  \global\bbl@csarg\chardef{wdir@#1}\z@
  \fi\fi%
\ifodd\bbl@engine
  \directlua{ Babel.locale_props\[	he\localeid\].textdir = 'l' }%
  \or
  \directlua{ Babel.locale_props\[	he\localeid\].textdir = 'r' }%
  \or
  \directlua{ Babel.locale_props\[	he\localeid\].textdir = 'al' }%
  \fi}%
\def\bbl@switchdir{%\bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%\bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%\bbl@exp{\\bbl@setdirs\\bbl@cl{wdir}}}%
\def\bbl@setdirs#1{%\ifcase\bbl@select@type % Strictly, not the right test\bbl@bodydir{#1}%\bbl@pardir{#1}<- Must precede \bbl@textdir\fi\bbl@textdir{#1}}%\iffodefbl@engine\%\bbl@csarg\chardef{wdir@#1}\one
  \directlua{ Babel.locale_props\[	he\localeid\].textdir = 'l' }%
  \or
  \directlua{ Babel.locale_props\[	he\localeid\].textdir = 'r' }%
  \or
  \directlua{ Babel.locale_props\[	he\localeid\].textdir = 'al' }%
  \fi}\def\bbl@textdir@i#1#2{\chardef\bbl@thetextdir\z@
  \@nameuse{setlatin}\\bbl@textdir@i\beginL\endL
  \else
  \chardef\bbl@thetextdir\@ne
  \@nameuse{setnonlatin}\\bbl@textdir@i\beginR\endR
  \fi\def\bbl@textdir@i#1#2{\chardef\bbl@thetextdir\z@
  \@nameuse{setlatin}\\bbl@textdir@i\beginL\endL
  \else
  \chardef\bbl@thetextdir\@ne
  \@nameuse{setnonlatin}\\bbl@textdir@i\beginR\endR
  \fi}%
% \iffodefbl@engine\%\bbl@csarg\chardef{wdir@#1}\one
  \directlua{ Babel.locale_props\[	he\localeid\].textdir = 'l' }%
  \or
  \directlua{ Babel.locale_props\[	he\localeid\].textdir = 'r' }%
  \or
  \directlua{ Babel.locale_props\[	he\localeid\].textdir = 'al' }%
  \fi%\iffedefbl@bidimode > 0:
  \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
  \DisableBabelHook{babel-bidi}
% Now the engine-dependent macros. TODO. Must be moved to the engine files.
\iffedefbl@bidimode > 0:
  \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
  \DisableBabelHook{babel-bidi}
% Now the engine-dependent macros. TODO. Must be moved to the engine files.
The following command is executed only if there is a right-to-left script (once). It activates the \everypar hack for xetex, to properly handle the par direction. Note text and par dirs are decoupled to some extent (although not completely).

\def\bbl@xeeverypar{\beginR\box\z@}
\section*{5.6 Local Language Configuration}

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

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

\section*{5.7 Language options}

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

Now, we set a few language options whose names are different from ldf files. These declarations are preserved for backwards compatibility, but they must be eventually removed. Use proxy files instead.
Another way to extend the list of 'known' options for babel was to create the file bblopts.cfg in which one can add option declarations. However, this mechanism is deprecated – if you want an alternative name for a language, just create a new .ldf file loading the actual one. You can also set the name of the file with the package option config=<name>, which will load <name>.cfg instead.

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

```
\bbl@foreach\bbl@language@opts{%
  \def\bbl@tempa{#1}%
  \ifx\bbl@tempa\bbl@opt@main\else
    \ifnum\bbl@iniflag<\tw@ % 0 ø (other = ldf)
      \bbl@ifunset{ds@#1}{%
        \DeclareOption{#1}{\bbl@load@language{#1}}%
        }%
    \else % + * (other = ini)
      \DeclareOption{#1}{%
        \bbl@ldfinit
        \babelprovide[import]{#1}%
        \bbl@afterldf{}}%
    \fi
  \fi}
\bbl@foreach\@classoptionslist{%
  \def\bbl@tempa{#1}%
  \ifx\bbl@tempa\bbl@opt@main\else
    \ifnum\bbl@iniflag<\tw@ % 0 ø (other = ldf)
      \bbl@ifunset{ds@#1}{%
        \IfFileExists{#1.ldf}{%
          \DeclareOption{#1}{\bbl@load@language{#1}}%
          }%
        }%
    \else % + * (other = ini)
      \IfFileExists{babel-#1.tex}{%
        \DeclareOption{#1}{%}
        \bbl@ldfinit
        \babelprovide[import]{#1}%
        \bbl@afterldf{}}%
    \fi
  \fi}
And we are done, because all options for this pass has been declared. Those already processed in the first pass are just ignored.
The options have to be processed in the order in which the user specified them (but remember class options are processes before):
```

\bbl@trace{Option 'main'}% 
\edef\bbl@opt@main{\@classoptionslist,\bbl@language@opts}
\let\bbl@tempc\@empty
\edef\bbl@templ{,\bbl@loaded,}
\edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
\bbl@for\bbl@tempb\bbl@tempa{%
  \edef\bbl@tempd{,\bbl@tempb,}%
  \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
  \bbl@xin{\bbl@tempd}{\bbl@templ}%
```

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

```
\bbl@trace{Option 'main'}%
\edef\bbl@opt@main{\@nnil}
\edef\bbl@classoptionslist{\@classoptionslist,\bbl@language@opts}
\let\bbl@empty\@empty
\edef\bbl@templ{,\bbl@loaded,}
\edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
\bbl@for\bbl@tempb\bbl@tempa{%
  \edef\bbl@tempd{,\bbl@tempb,}%
  \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
  \bbl@xin{\bbl@tempd}{\bbl@templ}%
```

92
In order to catch the case where the user didn’t specify a language we check whether \bbl@main@language, has become defined. If not, the nil language is loaded.

\ifx\bbl@main@language\@undefined
\bbl@info{% You haven’t specified a language as a class or package\%
option. I’ll load ‘nil’. Reported}
\bbl@load@language{nil}
\fi
\fi

⟨∗ kernel ⟩

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

The kernel of the babel system is currently stored in babel\,def. The file babel\,def contains most of the code. The file hyphen\,cfg is a file that can be loaded into the format, which is necessary when you want to be able to switch hyphenation patterns. Because plain \TeX\ users might want to use some of the features of the babel system too, care has to be taken that plain \TeX\ can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain \TeX\ and \LaTeX\, some of it is for the \LaTeX\ case only.

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

A proxy file for switch\,def

⟨ package ⟩
Loading hyphenation patterns

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

\let\bbl@onlyswitch\@empty
\input{babel.def}
\let\bbl@onlyswitch\@undefined
⟨/kernel⟩
⟨∗patterns⟩

Loading hyphenation patterns

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

\let\bbl@onlyswitch\@empty
\input{babel.def}
\let\bbl@onlyswitch\@undefined
⟨/kernel⟩
⟨∗patterns⟩

7 Loading hyphenation patterns

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

\let\bbl@onlyswitch\@empty
\input{babel.def}
\let\bbl@onlyswitch\@undefined
⟨/kernel⟩
⟨∗patterns⟩

Loading hyphenation patterns

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

\let\bbl@onlyswitch\@empty
\input{babel.def}
\let\bbl@onlyswitch\@undefined
⟨/kernel⟩
⟨∗patterns⟩

Loading hyphenation patterns

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

\let\bbl@onlyswitch\@empty
\input{babel.def}
\let\bbl@onlyswitch\@undefined
⟨/kernel⟩
⟨∗patterns⟩

Loading hyphenation patterns

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

\let\bbl@onlyswitch\@empty
\input{babel.def}
\let\bbl@onlyswitch\@undefined
⟨/kernel⟩
⟨∗patterns⟩
The macro \bbl@get@enc extracts the font encoding from the language name and stores it in \bbl@hyph@enc. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

Pattern files may contain assignments to \left@hyphenmin and \right@hyphenmin. \TeX does not keep track of these assignments. Therefore we try to detect such assignments and store them in the \langle lang⟩/hyphenmins macro. When no assignments were made we provide a default setting.

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

Then we globally store the settings of \left@hyphenmin and \right@hyphenmin 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\langle \langle language-name\rangle \} \}{ \langle number\rangle } \}{ \langle patterns-file\rangle } \}{ \langle exceptions-file\rangle }. Note the last 2 arguments are empty in ‘dialects’ defined in language.dat with ‘=’. Note also the language name can have encoding info.

Finally, if the counter \language is equal to zero we execute the synonyms stored.

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

\end{verbatim}

Now, hooks are defined. For efficiency reasons, they are dealt here in a special way. Besides \latex, format-specific configuration files are taken into account. \loadkernel currently loads nothing, but define some basic macros instead.
The configuration file can now be opened for reading.

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

\readconfigfile

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

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

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

Check for the end of the file. We must reverse the test for \ifeof without \else. Then reactivate the default patterns, and close the configuration file.

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

Also remove some macros from memory and raise an error if \toks@ is not empty. Finally load switch.def, but the latter is not required and the line inputting it may be commented out.
Font handling with fontspec

Add the bidi handler just before luatexload, 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 [mislplaced].

\begin{verbatim}
\DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
\DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101}
\DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102}
\DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201}
\DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202}
\DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203}
\end{verbatim}

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.

\begin{verbatim}
\bbl@trace{Font handling with fontspec}
\ifx\ExplSyntaxOn\@undefined\else
  \def\bbl@fs@warn@nx#1#2{% \bbl@tempfs is the original macro
    \in@{,#1,}{,no-script,language-not-exist,}%
    \ifin@\else\bbl@tempfs@nx{#1}{#2}\fi}
\fi
\end{verbatim}
\texttt{% For the default font, just in case:
\bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}\%
\expandafter\bbl@foreach\expandafter{\bbl@tempa}\%
(\bbl@csarg\edef{\bbl@tempb dflt@}{\{-\{\#2\}}% save bbl@rmdflt@
\bbl@exp\%
\\let<\bbl@tempb dflt@\languagename><\bbl@tempb dflt@\languagename>%
\\\bbl@foreach\bbl@font@fams{% ie bbl@rmdflt@lang / *scrt
(\bbl@csarg\def{\bbl@tempb dflt@##1}{\{-\{\#2\}})}%}
\fontfamily{\bbl@tempa}% Too!\texttt{\bbl@foreach\bbl@font@fams{%)
\global\let<\bbl@##1dflt@\languagename><\bbl@##1dflt@\languagename>%
\fontfamily{\bbl@##1family}%
\selectfont}}%
\DeclareTextFontCommand{\text#1}{\bbl@##1family}}%}

If the family in the previous command does not exist, it must be defined. Here is how:
\def\bbl@providefam#1{%
\bbl@exp{%
\\newcommand<#1default>{}% Just define it
\\bbl@add@list\\bbl@font@fams{#1}%
\\DeclareRobustCommand<#1family>{%\not@math@alphabet<#1family>\relax
% \prepare@family@series@update{#1}<#1default>% TODO. Fails
\fontfamily{#1default}\
\ifx>\UseHooks\@undefined\else\UseHook{#1family}\fi%}
\\selectfont}
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% 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{%\n          \bbl@ifunset{bbl@##1dflt@}{% Flag
            \\bbl@csarg\gdef{WFF@\f@family}{}% Flag
            \bbl@exp{\\\bbl@add\\bbl@tempa{* \<##1family>= \f@family\\% \space\space\fontname\font\\}}% \\
          \bbl@csarg\xdef{##1dflt@}{\f@family}%
          \expandafter\xdef\csname ##1default\endcsname{\f@family}}%}{}}%
        \ifx\bbl@tempa\@empty% There is nothing intrinsically wrong with it, but\%
          \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 % TODO - next should be global?, but even local does its job. I'm
\fi
```

Now the macros defining the font with fontspec.

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

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

```
\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{% 'Unprotected' macros return prev values
    \def\#2[#1]{% eg, \rmdefault{\bbl@rmdflt@lang}
    \bbl@exp{\\\\bbl@ifsamestring{\#2}[\f@family]%
      \bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}{%}
      \let\bbl@tempa\relax%
      {}}%}
  \% T0D0 - next should be global?, but even local does its job. I'm
  \% still not sure -- must investigate:
  \\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
```
\let\bbl@tempe\bbl@mapselect
\def\bbl@tempmb{\bbl@stripslash#4/}% Catcodes hack (better pass it).
\bbl@exp{\\bbl@replace\\bbl@tempmb{\bbl@stripslash\family/}{}}%
\let\bbl@mapselect\relax
\let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
\let#4\@empty % Make sure \renewfontfamily is valid
\bbl@exp{\
\let\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
\<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
\<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
\let\\bbl@font@rst#1#2#3#4{%
\bbl@csarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}
\let\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.

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

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

(⟨⟨Font selection⟩⟩)
Now, the code.
Support for interchar

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

\begin{enumerate}
\item \texttt{\edef{\xetexclass{default}}{\z@}}
\item \texttt{\edef{\xetexclass{cjkideogram}}{\one@}}
\item \texttt{\edef{\xetexclass{cjkleftpunctuation}}{\two@}}
\item \texttt{\edef{\xetexclass{cjkrightpunctuation}}{\three@}}
\item \texttt{\edef{\xetexclass{boundary}}{4095}}
\item \texttt{\edef{\xetexclass{ignore}}{4096}}
\end{enumerate}

The machinery is activated with a hook (enabled only if actually used). Here \texttt{\bb@tempc} is pre-set with \texttt{\usingxetex}, defined below. The standard mechanism based on \texttt{\originalTeX} to save, set and restore values is used. \texttt{\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.

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

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

\begin{verbatim}
\newcommand\babelinterchar[5][]{% 
  \let\bbl@kv@label\@empty 
  \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}% 
  \@namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 @empty}{% 
    \ifnum\language=\l@nohyphenation 
    \expandafter\@gobble 
    \else 
    \expandafter\@firstofone 
    \fi 
    {#5}}% 
  \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone 
  \fi 
  \}{}
\end{verbatim}

\begin{verbatim}
\DeclareRobustCommand\enablelocaleinterchar[1]{% 
  \bbl@ifunset{bbl@ic@#1@\languagename}{}{\bbl@error{#1 for 'languagename' cannot be enabled.\
    Maybe there is a typo.}}% 
  \bbl@csarg\let{ic@#1@\languagename}\@firstofone 
}\end{verbatim}

\begin{verbatim}
\providecommand\bbl@provide@intraspace{} 
\bbl@trace{Redefinitions for bidi layout} 
\def\bbl@sspre@caption{% 
  \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}} 
\ifx\bbl@opt@layout\@nnil\else % if layout=.. 
  \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi} 
  \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi} 
  \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi= 
  \def\@hangfrom#1{% 
    \setbox\@tempboxa\hbox{{#1}}% 
    \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi 
    \noindent\box\@tempboxa} 
  \def\raggedright{% 
  \XeTeXinterchartoks 
  \bbl@ifunset{bbl@xeinterclass@\bbl@tempa \bbl@tempb}{% 
    \bbl@error{#1 for 'languagename' cannot be enabled.\% 
      Maybe there is a typo.}}% 
  \bbl@ifunset{bbl@xeinterclass@\bbl@tempa \bbl@tempb}{% 
    \bbl@error{#1 for 'languagename' cannot be disabled.\% 
      Maybe there is a typo.}}% 
  \bbl@csarg\let{ic@\bbl@kv@label @\languagename}\@firstofone 
}\end{verbatim}

\section{10. 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: \texttt{\adim\bbl@startskip,} \texttt{\advance\bbl@startskip\adim,} \texttt{\bbl@startskip\adim,} \texttt{\bbl@startskip\adim.}

Consider \texttt{xetex babel} as a shorthand for \texttt{tex–xet babel}, 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.

10.3 LuaTeX

The loader for luatex is based solely on language.dat, which is read on the fly. The code shouldn’t be executed when the format is build, so we check if \AddBabelHook is defined. Then comes a modified version of the loader in hyphen.cfg (without the hyphenmins stuff, which is under the direct control of babel).
The names `\@<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, lua(e)tex is taken into account. Formerly, loading of patterns on the fly didn't work in this format, but with the new loader it does. Unfortunately, the format is not based on babel, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format language.dat is used (under the principle of a single source), instead of language.def.

Of course, there is room for improvements, like tools to read and reassign languages, which would require modifying the language list, and better error handling. We need catcode tables, but no format (targeted by babel) provide a command to allocate them (although there are packages like tablestack). FIX - This isn’t true anymore. For the moment, a dangerous approach is used - just allocate a high random number and cross the fingers. To complicate things, etex.sty changes the way languages are allocated.

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

5086 \langle\text{lualtex}\rangle
5087 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
5088 \bbl@trace{Read language.dat}
5089 \ifx\bbl@readstream\@undefined
5090 \csname newread\endcsname\bbl@readstream
5091 \fi
5092 \begingroup
5093 \toks@{}
5094 \count@\z@ % 0=start, 1=0th, 2=normal
5095 \def\bbl@process@line#1#2 #3 #4 {%
5096 \ifequiv\%1%
5097 \bbl@process@synonym{#2}\
5098 \else
5099 \bbl@process@language{#1#2}{#3}{#4}\
5100 \fi
5101 \ignorespaces
5102 \def\bbl@manylang{%
5103 \ifnum\count@>\@ne
5104 \bbl@info{Non-standard hyphenation setup}\
5105 \fi
5106 \ignorespaces}
5107 \def\bbl@manylang{%
5108 \ifnum\bbl@last>\@ne
5109 \bbl@info{Non-standard hyphenation setup}\
5110 \fi
5111 \let\bbl@manylang\relax}
5112 \def\bbl@process@language#1#2#3#4{%
5113 \ifcase\count@
5114 \@ifundefined{zth@#1}\{\count@\tw@}\{\count@\@ne}\
5115 \or
5116 \count@\tw@\
5117 \fi
5118 \ifnum\count@=\tw@\
5119 \expandafter\addlanguage\csname l@#1\endcsname
5120 \language\allocationnumber\
5121 \chardef\bbl@last\allocationnumber\
5122 \bbl@manylang\
5123 \let\bbl@elt\relax
5124 \xdef\bbl@languages{%
5125 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
\def\bbl@process@synonym@aux#1#2{% 
\global\expandafter\chardef\csname l@#1\endcsname#2\relax 
\let\bbl@elt\relax 
\xdef\bbl@languages{\bbl@languages\bbl@elt{#1}{#2}{}}% 
\def\bbl@process@synonym#1{% 
\ifcase\count@ 
\toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}% 
or 
@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}% 
else 
\bbl@process@synonym@aux{#1}{\the\bbl@last}{}% 
\fi} 
\ifx\bbl@languages\@undefined % Just a (sensible?) guess 
\chardef\l@english\z@ 
\chardef\l@USenglish\z@ 
\chardef\bbl@last\z@ 
\global\@namedef{bbl@hyphendata@0}{{hyphen.tex}{}} 
\gdef\bbl@languages{\bbl@elt{english}{0}{hyphen.tex}{}% 
\bbl@elt{USenglish}{0}{}{}}% 
\else 
\global\let\bbl@languages@format\bbl@languages 
\def\bbl@elt#1#2#3#4{% Remove all except language 0 
\ifnum#2>\z@\else 
\noexpand\bbl@elt{#1}{#2}{#3}{#4}% 
\fi} 
\xdef\bbl@languages{\bbl@languages} 
\def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}}% Define flags 
\bbl@languages 
\openin\bbl@readstream=language.dat 
\ifeof\bbl@readstream 
\bbl@warning{I couldn't find language.dat. No additional\% 
patterns loaded. Reported}% 
\else 
\loop 
\endlinechar\m@ne 
\read\bbl@readstream to \bbl@line 
\endlinechar`\^^M 
\if T\ifeof\bbl@readstream F,fi T,relax 
\ifx\bbl@line\empty\else 
\edef\bbl@line{\bbl@line\space\space\space}% 
\expandafter\bbl@process@line\bbl@line\relax 
\fi 
\fi 
\repeat 
\closein\bbl@readstream 
\endgroup 
\bbl@trace{Macros for reading patterns files} 
\def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}} 
\ifx\babelcatcodetable\@undefined 
\ifx\newcatcodetable\@undefined 
\def\babelcatcodetable{5211} 
\def\bbl@pattcodes{\numexpr\babelcatcodetable+1\relax} 
\else 
\newcatcodetable\babelcatcodetable 
\newcatcodetable\bbl@pattcodes 
\fi 
\else
\endinput

\begin{verbatim}
5247 \csname bbl@HyphenData@the\language\endcsname}
5248 \endinput
\fi
5249 % Here stops reading code for hyphen.cfg
5250 % The following is read the 2nd time it's loaded
5251 \begin{group}\% T000 - to a lua file
5252 \catcode`\%=12
5253 \catcode`\'=12
5254 \catcode`\":12
5255 \catcode`\:=12
5256 \directlua{
5257 Babel = Babel or {}
5258 function Babel.bytes(line)
5259 return line:gsub("(.)",
5260 function (chr) return unicode.utf8.char(string.byte(chr)) end)
5261 end
5262 function Babel.begin_process_input()
5263 if luatexbase and luatexbase.add_to_callback then
5264  luatexbase.add_to_callback('process_input_buffer',
5265  Babel.bytes,'Babel.bytes')
5266 else
5267  Babel.callback = callback.find('process_input_buffer')
5268  callback.register('process_input_buffer',Babel.bytes)
5269 end
5270 end
5271 function Babel.end_process_input()
5272 if luatexbase and luatexbase.remove_from_callback then
5273  luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
5274 else
5275  callback.register('process_input_buffer',Babel.callback)
5276 end
5277 end
5278 function Babel.addpatterns(pp, lg)
5279 local lg = lang.new(lg)
5280 local pats = lang.patterns(lg) or ''
5281 lang.clear_patterns(lg)
5282 for p in pp:gmatch('[^%s]+') do
5283 ss = ''
5284 for i in string.utfcharacters(p:gsub('%d', '')) do
5285 ss = ss .. '%d?' .. i
5286 end
5287 ss = ss:gsub('%%d%?%.', '%%.') .. '%d?'
5288 ss = ss:gsub('%.%%d%?', '%%')
5289 pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5290 if n == 0 then
5291 tex.sprint(
5292 \[
5293 \csname bbl@info\endcsname{New pattern: %]
5294 .. p .. \[
5295 ]
5296 pats = pats .. ' ' .. p
5297 else
5298 tex.sprint(
5299 \[
5300 \csname bbl@info\endcsname{Renew pattern: %]
5301 .. p .. \[
5302 ]
5303 pats = pats .. ' ' .. p
5304 end
5305 lang.patterns(lg, pats)
5306 end
5307 Babel.characters = Babel.characters or {}
5308 Babel.ranges = Babel.ranges or {}
5309 function Babel.hlist_has_bidi(head)
5310 local has_bidi = false
5311 local ranges = Babel.ranges
5312 for item in node.traverse(head) do
5313 if item.id == node.id'glyph' then
5314 \end{verbatim}
local itemchar = item.char
local chardata = Babel.characters[itemchar]
local dir = chardata and chardata.d or nil
if not dir then
    for nn, et in ipairs(ranges) do
        if itemchar < et[1] then
            break
        elseif itemchar <= et[2] then
            dir = et[3]
            break
        end
    end
    if dir and (dir == 'al' or dir == 'r') then
        has_bidi = true
    end
end
return has_bidi
end

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

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

\if\newattribute\@undefined\else % Test for plain
\newattribute\bbl@attr@locale
\directlua{ Babel.attr_locale = luatexbase.registernumber'\bbl@attr@locale' }
\AddBabelHook{luatex}{beforeextras}{ \setattribute\bbl@attr@locale\localeid}
\fi
\def\BabelStringsDefault{unicode}
\let\luabbl@stop\relax
\AddBabelHook{luatex}{encodedcommands}{\def\bbl@tempa{utf8}\def\bbl@tempb{#1}\ifx\bbl@tempa\bbl@tempb\else
\directlua{Babel.begin_process_input()}%\def\luabbl@stop{\directlua{Babel.end_process_input()}}\fi}
\AddBabelHook{luatex}{stopcommands}{\luabbl@stop\let\luabbl@stop\relax}\AddBabelHook{luatex}{patterns}{\@ifundefined{bbl@hyphendata@\the\language}{\def\bbl@elt##1##2##3##4{\ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
\def\bbl@tempa{utf8}\def\bbl@tempb{#1}\ifx\bbl@tempa\bbl@tempb\else
\directlua{Babel.begin_process_input()}%\def\luabbl@stop{\directlua{Babel.end_process_input()}}\fi}%}

\AddBabelHook{luatex}{stopcommands}{\luabbl@stop
\let\luabbl@stop\relax}
\AddBabelHook{luatex}{patterns}{\ifundefined{bbl@hyphendata@\the\language}{\def\bbl@elt##1##2##3##4{\ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...\def\bbl@tempa{utf8}\def\bbl@tempb{#1}\ifx\bbl@tempa\bbl@tempb\else
\directlua{Babel.begin_process_input()}%\def\luabbl@stop{\directlua{Babel.end_process_input()}}\fi}%}
10.4 Southeast Asian scripts

First, some general code for line breaking, used by \texttt{\string\babelposthyphenation}.
Replace regular (i.e., implicit) discretionary spaces skips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionary are not touched. See Unicode UAX14.

```lua
% TODO - to a lua file
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

Babel.intraspaces = Babel.intraspaces or {
    \csname bbl@sbcp@\languagename\endcsname\} = {
b = #1, p = #2, m = #3
    Babel.locale_props[\the\localeid].intraspace = {
b = #1, p = #2, m = #3
    }
}
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
    }
}
\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
}
}\begingroup
\catcode\%=12
\catcode\^=14
\catcode\'=12
\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
        local quad = 655360 ^% 10 pt = 655360 = 10 * 65536
        for item in node.traverse(head) do
            local i = item.id
```

114
if i == node.id'glyph' then
    last_char = item
else if 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, Cyril1
            local intraspaces = Babel.intraspaces[lg]
            local intrapenalties = Babel.intrapenalties[lg]
            local n
            if intrapenalty ~= 0 then
                n = node.new(l4, 0) ^% penalty
                n.penalty = intrapenalty
                node.insert_before(head, item, n)
            end
            n = node.new(l2, l3) ^% (glue, spaceskip)
            node.setglue(n, intraspaces.b * quad,
                    intraspaces.p * quad,
                    intraspaces.m * quad)
            node.insert_before(head, item, n)
            node.remove(head, item)
        end
    end
end
end
end
}

\bb@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
\gdef\bb@cjkintraspace{%
  \let\bb@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]
          local class = Babel.cjk_class[item.char].c
          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, Cyril1
                local intraspaces = Babel.intraspaces[lg]
                local intrapenalties = Babel.intrapenalties[lg]
                local n
                if intrapenalty ~= 0 then
                    n = node.new(l4, 0) ^% penalty
                    n.penalty = intrapenalty
                    node.insert_before(head, item, n)
                end
                n = node.new(l2, l3) ^% (glue, spaceskip)
                node.setglue(n, intraspaces.b * quad,
                        intraspaces.p * quad,
                        intraspaces.m * quad)
                node.insert_before(head, item, n)
                node.remove(head, item)
            end
        end
    end
end
end

115
if props.cjk_quotes and props.cjk_quotes[item.char] then
    class = props.cjk_quotes[item.char]
end

if class == 'cp' then class = 'cl' end

if class == 'id' then class = 'I' end

local br = 0

if class and last_class and Babel.cjk_breaks[last_class][class] then
    br = Babel.cjk_breaks[last_class][class]
end

if br == 1 and props.linebreak == 'c' and
    lang ~= \the\l@nohyphenation\space and
    last_lang ~= \the\l@nohyphenation then
    local intrapenalty = props.intrapenalty
    if intrapenalty ~= 0 then
        local n = node.new(14, 0) % penalty
        n.penalty = intrapenalty
        node.insert_before(head, item, n)
    end
    local intraspace = props.intraspace
    local n = node.new(12, 13) % (glue, spaceskip)
    node.setglue(n, intraspace.b * quad,
                 intraspace.p * quad,
                 intraspace.m * quad)
    node.insert_before(head, item, n)
end

if font.getfont(item.font) then
    quad = font.getfont(item.font).size
end

last_class = class
last_lang = lang
else % if penalty, glue or anything else
    last_class = nil
end

lang.hyphenate(head)

end

\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
if Babel.sea_enabled then
    Babel.sea_disc_to_space(head)
end

end
\endgroup
\def\bbl@provide@intraspace{%\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
  \bbl@xin{/c}{/\bbl@cl{lnbrk}}%\ifin@%cjk
    \bbl@cjkintraspace
    \directlua{
      Babel = Babel or {}
      Babel.locale_props = Babel.locale_props or {}
      Babel.locale_props[\the\localeid].linebreak = 'c'
    }%\bbl@exp{\bbl@intraspace\bbl@cl{intsp}\@@}
    \ifx\bbl@KVP@intrapenalty\@nnil
      \bbl@intrapenalty0\@@
    \fi%sea
  \else
    \bbl@seaintraspace
    \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\@@}
    \directlua{
      Babel = Babel or {}
      Babel.sea_ranges = Babel.sea_ranges or {}
      Babel.set_chranges('\bbl@cl{sbcp}',
        \bbl@cl{chrng})
    }%\ifx\bbl@KVP@intrapenalty\@nnil
      \bbl@intrapenalty0\@@
    \fi%sea
  \else
    \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
  \fi}
\def\bblar@chars{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{0626,0628,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%0640,0641,0642,0643,0644,0645,0646,0647,0648,0649}
\def\bblar@longated{%0626,0628,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,0634,0635,0636,0638,063A,063B,063C,063D,063E,063F,0640,0641,0642,0643,0644,0645,0646,0647,0648,0649}
\begin{group}
\catcode`_=11 \catcode`:11
\gdef\bblar@nofswarn{\gdef\msg_warning:nnx##1##2##3{}}
\endgroup
\def\bbl@arabicjust{% TODO. Allow for several locales.
\let\bbl@arabicjust\relax
\newattribute\bblar@kashida
\directlua{\Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%\bblar@kashida=\z@
\bbl@patchfont{{\bbl@parsejalt}}%\directlua{
10.6 Arabic justification
WIP. \bbl@arabicjust is executed with both elongated an kashida. This must be fine tuned. The
attribute kashida is set by transforms with kashida-
Babel.arabic.elong_map = Babel.arabic.elong_map or {}
Babel.arabic.elong_map[localeid] = {}
luatexbase.add_to_callback('post_linebreak_filter',
        Babel.arabic.justify, 'Babel.arabic.justify')
luatexbase.add_to_callback('hpack_filter',
        Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
}

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

\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{\%
            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
        }\%
    }\%
}\bbl@parsejalt{%\bf@exp{\bbl@foreach{#1}}{%\bbl@ifunset{bblar@JE@##1}\%
    \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
    \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
% \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
% \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
% \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
% \addfontfeature{RawFeature=+jalt}%
\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[localeid][fontid] = {}
            tex.print([[\string\csnamebbl@parsejalti\endcsname]])
        end
    end
\endgroup
\gdef\bbl@parsejalti{%
    \begingroup
        \let\bbl@parsejalt\relax % To avoid infinite loop
    \edef\bbl@tempf{\fontid\font}\%
    \bblar@mofswarm
    \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
    \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
    \addfontfeature{RawFeature=+jalt}%
    % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
    \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[localeid][\bbl@tempf] = Babel.arabic.dest[k]
            end
        end
    \endgroup
\gdef\bbl@parsejalt% The actual justification (inspired by CHICKENIZE).
\begingroup
\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(node.id'\hlist', head) do
    Babel.arabic.justify_hlist(head, line)
end
return head
end

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

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

    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
        ends = {} % Stores elongated candidates of each line
        k_list = {} % And all letters with kashida
        pos_inline = 0 % Not yet used

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

end
\endgroup
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
    if width > goal then
        curr.char = oldchar
        break
    end
    table.remove(elongs, x)
end

% == Tatwil ==
if #k_list == 0 then goto next_line end
width = node.dimensions(line.head) % The 'natural' width
k_curr = #k_list % Traverse backwards, from the end
wt_pos = 1
while width < goal do
    subst_done = true
    k_item = k_list[k_curr].node
    if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
        d = node.copy(k_item)
        d.char = 0x0640
        d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
        d.xoffset = 0
        line.head, new = node.insert_after(line.head, k_item, d)
        width_new = node.dimensions(line.head)
        if width > goal or width == width_new then
            break
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
end
% Limit the number of tatweel by removing them. Not very efficient,
% but it does the job in a quite predictable way.
if Babel.arabic.kashida_limit > -1 then
cnt = 0
for n in node.traverse_id(GLYPH, line.head) do
if n.char == 0x0640 then
cnt = cnt + 1
if cnt > Babel.arabic.kashida_limit then
node.remove(line.head, n)
end
else
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
end % if process line
end
⟨⟨
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 a 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}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
{0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
['Armn'] = {{0x0530, 0x058F}},
['Beng'] = {{0x0900, 0x09FF}},
['Copt'] = {{0x0382, 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},
{0x20000, 0x2A6DF}, {0x2A700, 0x2B73F}, {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
['Hebr'] = {{0x0590, 0x05FF}},
['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
{0xA960, 0xA97F}, {0xAC00, 0xD7AF}, {0xFF00, 0xFFEF}},
['Kana'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
['Khmr'] = {{0x0C80, 0x0CFF}},
['Kore'] = {{0x0E00, 0x0E7F}},
['Laoo'] = {{0x0E00, 0x0EFF}},
['Latn'] = {{0x0000, 0x007F}, {0xA720, 0xA77F}, {0x0860, 0x086F}},
['Mahj'] = {{0x11150, 0x1117F}},
['Mlym'] = {{0x0D00, 0x0D7F}},
['Mymr'] = {{0x0D00, 0x0D7F}},
['Orya'] = {{0x0E00, 0x0E7F}},
['Sinh'] = {{0x0D00, 0x0D7F}},
['Syr'] = {{0x0E00, 0x0E7F}},
['Tam'] = {{0x0E00, 0x0E7F}},
['Telu'] = {{0x0C00, 0x0C7F}},
['Tfng'] = {{0x2D30, 0x2D7F}},
['Thai'] = {{0x0E00, 0x0E7F}},
['Tib'] = {{0x0F00, 0x0FF0}},
['Vaii'] = {{0x0A00, 0x0A63F}},
['Yi'] = {{0x0A00, 0x0A48F}, {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')

122
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
      % 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
    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 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) \string~= 11 then
      toloc = nil
    end
    if toloc and Babel.locale_props[toloc].script
      and Babel.locale_props[node.get_attribute(item, LOCALE)].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
    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)
  else
    inmath = (item.subtype == 0)
  end
end
The code for \babelcharproperty is straightforward. Just note the modified lua table can be
different.

\newcommand\babelcharproperty[1]{\%\count@=#1\relax\ifvmode\expandafter\bbl@chprop\else\bbl@error{\string\babelcharproperty\space can be used only in vertical mode (preamble or between paragraphs)}\%(See the manual for further info)\%\fi}\newcommand\bbl@chprop[3]\[3\]{\@tempcnta=#1\relax\bbl@ifunset{bbl@chprop@#2}{\bbl@error{No property named ‘#2’. Allowed values are direction (bc), mirror (bmg), and linebreak (lb)}\%(See the manual for further info)}\{}\def\bbl@chprop@direction#1{\directlua{Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}Babel.characters[\the\count@][‘d’] = ‘#1’}}\let\bbl@chprop@bc=\bbl@chprop@direction\def\bbl@chprop@mirror#1{\directlua{Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}Babel.characters[\the\count@][‘m’] = ‘\number#1’}}\let\bbl@chprop@bmg=\bbl@chprop@mirror\def\bbl@chprop@linebreak#1{\directlua{Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}Babel.cjk_characters[\the\count@][‘c’] = ‘#1’}}\let\bbl@chprop@lb=\bbl@chprop@linebreak\def\bbl@chprop@locale#1{\directlua{Babel.chr_to_loc = Babel.chr_to_loc or {}Babel.chr_to_loc[\the\count@] = \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space}}}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.

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

\begin{verbatim}
\begin{verbatim}
\catcode `\-=12
\catcode `\%=12
\catcode `\&=14
\catcode `\|=12
\gdef\babelprehyphenation{%
  \ifnextchar[{}{\bbl@settransform{0}}{\bbl@settransform{0}\[\]}}
\gdef\babelposthyphenation{%
  \ifnextchar[{}{\bbl@settransform{1}}{\bbl@settransform{1}\[\]}}
\gdef\bbl@settransform#1[#2]#3#4#5{%
  \ifcase#1
    \bbl@activateprehyphen
    \or
    \bbl@activateposthyphen
  \fi
  \begingroup
    \def\babeltempa{\bbl@add@list\babeltempb}&%
    \let\babeltempb\@empty
    \def\bbl@tempa{#5}&%
    \bbl@replace\bbl@tempa{,}{ ,}&%
    \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
      \bbl@ifsamestring{##1}{remove}{\bbl@add@list\babeltempb{nil}}&%
      \directlua{
        local rep = \[=\[##1\]=
        rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
        rep = rep:gsub('^%s*(insert)%s*$', 'insert = true ,')
        rep = rep:gsub('^%s*(string)%s*=%s*(\[^%s,\]*)', Babel.capture_func)
        if #1 == 0 or #1 == 2 then
          rep = rep:gsub('^%s*(space)%s*=%s*\([%d.\]+\)+\([%d.\]+\)+\([%d.\]+\)', 'space = { , %2, %3, %4 ... }')
        end
        rep = rep:gsub('^%s*(spacefactor)%s*=%s*\([%d.\]+\)+\([%d.\]+\)+\([%d.\]+\)', 'spacefactor = { , %2, %3, %4 ... }')
        rep = rep:gsub('^%s*(kashida)%s*=%s*\([-%s,\]*)', Babel.capture_kashida)
        else
          rep = rep:gsub( '^(no)%s*=%s*\([-%s,\]*)', Babel.capture_func)
          rep = rep:gsub( '^(pre)%s*=%s*\([-%s,\]*)', Babel.capture_func)
          rep = rep:gsub( '^(post)%s*=%s*\([-%s,\]*)', Babel.capture_func)
        end
        tex.print{\[
          \string\babeltempa{\[\] .. rep .. \[\]}}
      }}&%
      \bbl@foreach\babeltempb{&%
        \bbl@forkv{##1}{&%
          \in@{,####1,}{,nil,step,data,remove,insert,string,no,pre,}&%
          \no,post,penalty,kashida,space,spacefactor,}&%
        \ifin@else
          \bbl@error
          {Bad option '####1' in a transform.\&%
            I'll ignore it but expect more errors}\&%
          {See the manual for further info.}\&%
        \fi}
      }&%
      \let\bbl@kv@attribute\relax
      \let\bbl@kv@label\relax
      \let\bbl@kv@fonts\@empty
      \bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}&%
      \ifx\bbl@kv@fonts\@empty\else\bbl@settransform\fi
      \ifx\bbl@kv@attribute\relax
        \ifx\bbl@label\relax
          \bbl@exp{\[
            \bbl@trim\def{\bbl@kv@fonts{\bbl@kv@fonts}}\&%
          }&%
        \else
          \bbl@exp{\[
            \bbl@trim\def{\bbl@kv@attribute{\bbl@kv@attribute}}\&%
          }&%
        \fi
      \fi
    \endgroup
  \endgroup
}\endgroup
\end{verbatim}
\end{verbatim}
\end{verbatim}
\edef\bbl@kv@attribute{\bbl@ATR@\bbl@kv@label @#3@\bbl@kv@fonts}
\count@\z@
\def\bbl@elt##1##2##3{&%
\bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
{\count@@ne}&%
{\bbl@error
(Transforms cannot be re-assigned to different fonts. The conflict is in \'\bbl@kv@label\'.
Apply the same fonts or use a different label)&
(See the manual for further details,))&%
{}&%
\bbl@transfont@list
\ifnum\count@=\z@
 {\bbl@exp{\global\\bbl@add\\bbl@transfont@list
{\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}}&%
\fi
\bbl@ifunset{\bbl@kv@attribute}&%
{\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
{}&%
\global\bbl@carg\setattribute{\bbl@kv@attribute}@ne
\else
\edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
\fi
\directlua{
local lbkr = Babel.linebreaking.replacements[#1]
local u = unicode.utf8
local id, attr, label
if #1 == 0 then
 id = \the\csname bbl@id@@#3\endcsname \space
else
 id = \the\csname l@#3\endcsname \space
end
\if\bbl@kv@attribute\relax
 attr = -1
 \else
 attr = luatexbase.registernumber'\bbl@kv@attribute'
 \fi
\if\bbl@kv@label\relax\else \&% Same refs:
 label = [==[\bbl@kv@label]==]
 \fi
\&% Convert pattern:
local patt = string.gsub([==[#4]==], '%s', '')
if #1 == 0 then
 patt = string.gsub(patt, '()', '')
end
if not u.find(patt, '()', nil, true) then
 patt = '()' .. patt .. '()'
end
if #1 == 1 then
 patt = string.gsub(patt, '%(%)%^', '^()')
patt = string.gsub(patt, '%$%(%)', '()$')
end
patt = u.gsub(patt, '{(.)}',
function (n)
 return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
end)
patt = u.gsub(u.char(tonumber(n, 16)), '%p', '%%%1')
end)
The following experimental (and unfinished) macro applies the prehyphenation transforms for the current locale to a string (characters and spaces) and processes it in a fully expandable way (among
other limitations, the string can’t contain 
==
). The way it operates is admittedly rather cumbersome: it converts the string to a node list, processes it, and converts it back to a string. The lua code is in the lua file below.

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

10.9 Bidi

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

\def\bbl@activate@preotf{% \let\bbl@activate@preotf\relax % only once \directlua{ % Babel = Babel or {} % function Babel.pre_otfload_v(head) % if Babel.numbers and Babel.digits_mapped then % head = Babel.numbers(head) % end % if Babel.bidi_enabled then % head = Babel.bidi(head, false, dir) % end % return head % 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 \mathboxdir hack is activated every math with the package option bidi=.

\breakafterdirmode=1 \ifnum\bbl@bidimode=1\textbf{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\bbl@bidimode\or % require('babel-bidi-basic.lua') %}
\or
\newattribute{bbl@attr@dir}
\directlua{ Babel.attr_dir = luatexbase.registernumber\'bbl@attr@dir' }
\bbl@exp{\output\{bodydir\pagedir\the\output\}}
\fi
\chardef{bbl@thetextdir}@\z@
\chardef{bbl@thepardir}@\z@
\def{bbl@getluadir}{%#1}{
  \directlua{
    if tex.#1dir == 'TLT' then
      tex.sprint(\'0\')
    elseif tex.#1dir == 'TRT' then
      tex.sprint(\'1\')
    end}
}
\def{bbl@setluadir}{%1#2#3}{%#1\textdir#2\textdir#3}{
  \ifcase#3\relax
    \ifcase{bbl@getluadir}{#1}\relax
      #2 TLT\relax
    \else
      \ifcase{bbl@getluadir}{#1}
        #2 TRT\relax
      \fi
    \fi}
  \else
    \ifcase{bbl@getluadir}{#1}
      #2 TLT\relax
    \else
      #2 TRT\relax
    \fi
  \fi}
\chardef{bbl@thedir}@\z@
\def{bbl@textdir}{#1}{
  bbl@setluadir{text}\textdir{#1}%
  \chardef{bbl@thetextdir}@\relax
  \edef{bbl@thedir}{\the\numexpr{bbl@thepardir}*4+#1}%
  \setattribute{bbl@attr@dir}{\numexpr{bbl@thepardir}*4+#1}}
\def{bbl@pardir}{#1}{
  bbl@setluadir{par}\pardir{#1}%
  \chardef{bbl@thepardir}@\relax}
\def{bbl@bodydir}{bbl@setluadir{body}\bodydir}% Unused once
\def{bbl@pagedir}{bbl@setluadir{page}\pagedir}% Unused once
\def{bbl@dirparastext}{\pardir\the\textdir\relax}% Used once

RTL text inside math needs special attention. It affects not only to actual math stuff, but also
to 'tabular', which is based on a fake math.
\ifnum{bbl@bidimode}>\z@ \& Any bidi=
  \def{bbl@insidecmath}{0}%
  \def{bbl@everymath}{\def{bbl@insidecmath}{1}}
  \def{bbl@everydisplay}{\def{bbl@insidecmath}{2}}
  \frozen@everymath\expandafter{%
    \expanded@bbl@everymath\expanded@%}
  \frozen@everydisplay\expandafter{%
    \expanded@bbl@everydisplay\expanded@%}
\AtBeginDocument{
  \directlua{
    function Babel.math_box_dir(head)
      if not (token.get_macro\('bbl@insidecmath'\) == '\0') then
        if Babel.hlist_has_bidi\(head\) then
          local d = node.new\(node.id'dir'\)
          d.dir = '+TRT'
          node.insert_before\(head, node.has_glyph\(head\), d)\)
          for item in node.traverse\(head\) do
            node.set_attribute\(item, Babel.attr_dir, token.get_macro\('bbl@thedir'\))
          end
        end
    end
  end}

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 \texttt{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 \texttt{amsmath}) the nightmare began. I'm still not sure how \texttt{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 \texttt{math} (II) nodes too).

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

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

With the issue \#15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, \texttt{tabular} seems to work (at least in simple cases) with \texttt{array}, \texttt{tabularx}, \texttt{hhline}, \texttt{colortbl}, \texttt{longtable}, \texttt{booktabs}, etc. However, \texttt{dcolumn} still fails.
function Babel.numbers(head) 
  local LOCALE = Babel.attr_locale 
  local GLYPH = node.id'glyph' 
  local inmath = false 
  for item in node.traverse(head) do 
    if not inmath and item.id == GLYPH then 
      local temp = node.get_attribute(item, LOCALE) 
      if Babel.digits[temp] then 
        local chr = item.char 
        if chr > 47 and chr < 58 then 
          item.char = Babel.digits[temp][chr-47] 
        end 
      end 
    elseif item.id == node.id'math' then 
      inmath = (item.subtype == 0) 
    end 
  end 
  return head 
end
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>\z@ % 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
    \@ifmmode%}

\begin{document}

\tableofcontents

\chapter{Introduction}

\section{Background}

\subsection{Motivation}

\section{Theoretical Framework}

\section{Methodology}

\chapter{Results}

\section{Discussion}

\chapter{Conclusion}

\backmatter

\bibliographystyle{plain}

\bibliography{references}

\end{document}
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.}}\}
\directlua{\luatexbase.add_to_callback("process_output_buffer",\Babel.discard_sublr,\"Babel.discard_sublr")}
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.

10.11 Lua: transforms

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: str_to_nodes converts the string returned by a function to a node list, taking the node at base as a model (font, language, etc.); fetch_word fetches a series of glyphs and discretionary, 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 latex manual), we must convert it to a utf8 position. With first, the last byte can be the leading byte in a utf8 sequence, so we just remove it and add 1 to the resulting length. With last we must take into account the capture position points to the next character. Here word_head points to the starting node of the text to be matched.
if fn == nil then return nil end
for s in string.utfvalues(fn(matches)) do
  if base.id == 7 then
    base = base.replace
  end
  n = node.copy(base)
  n.char = s
  if not head then
    head = n
  else
    last.next = n
  end
  last = n
end
return head

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

-- Merging both functions doesn't seem feasible, because there are too
-- many differences.
Babel.fetch_subtext[0] = function(head)
  local word_string = ''
  local word_nodes = {}
  local lang
  local item = head
  local inmath = false
  while item do
    if item.id == 11 then
      inmath = (item.subtype == 0)
    end
    if inmath then
      -- pass
    elseif item.id == 29 then
      local locale = node.get_attribute(item, Babel.attr_locale)
      local word_string = ''
      local word_nodes = {}
      local lang
      local item = head
      local inmath = false
      local inmath
      if locale then
        lang = locale
      else
        lang = node.lang
      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
      break
    end
    if item.id == 12 and item.subtype == 13 then
      word_string = word_string .. ' '
    end
    word_nodes[#word_nodes+1] = item
  end
  word_string = word_string .. Babel.us_char
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 -- 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
    else
        word_string = word_string .. Babel.us_char
        word_nodes[#word_nodes+1] = item -- Will be ignored
    end
item = item.next
end

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

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

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

Babel.us_char = string.char(31)

Babel.us_char = string.char(31)

function Babel.hyphenate_replace(head, mode)
local u = unicode.utf8
local lbkr = Babel.linebreaking.replacements[mode]
local word_head = head
while true do -- for each subtext block
local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
if Babel.debug then
print()
print((mode == 0) and '@@@@<' or '@@@@>', w)
end
if nw == nil and w == '' then break end
if not lang then goto next end
if not lbkr[lang] then goto next end
-- For each saved (pre|post)hyphenation. TODO. Reconsider how
-- loops are nested.
for k=1, #lbkr[lang] do
local p = lbkr[lang][k].pattern
local r = lbkr[lang][k].replace
local attr = lbkr[lang][k].attr or -1
if Babel.debug then
print('*****', p, mode)
end
-- This variable is set in some cases below to the first "byte"
-- after the match, either as found by u.match (faster) or the
-- computed position based on sc if w has changed.
local last_match = 0
local step = 0
-- For every match.
while true do
if Babel.debug then
print('=====')
end
local new -- used when inserting and removing nodes
local matches = { u.match(w, p, last_match) }
if #matches < 2 then break end

-- 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)
-- Non re-fetched substrings may contain \31, which separates
-- substrings.
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
  if Babel.debug then
    Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
  end
  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)
local crep = r[rc]
local item = w_nodes[sc]
local item_base = item
local placeholder = Babel.us_char
local d

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

if crep then
    step = crep.step or 0
end

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

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

142
7134   end -- for
7135   node.remove(head, item)
7136 end -- if ''
7137 last_match = utf8.offset(w, sc+1+step)
7138 goto next
7139
7140 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7141   d = node.new(7, 3) -- (disc, regular)
7142   d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7143   d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7144   d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7145   d.attr = item_base.attr
7146   if crep.pre == nil then -- TeXbook p96
7147     d.penalty = crep.penalty or tex.hyphenpenalty
7148   else
7149     d.penalty = crep.penalty or tex.exhyphenpenalty
7150   end
7151   placeholder = '|'
7152   head, new = node.insert_before(head, item, d)
7153
7154 elseif crep and crep.penalty then
7155   d = node.new(14, 0) -- (penalty, userpenalty)
7156   d.attr = item_base.attr
7157   d.penalty = crep.penalty
7158   head, new = node.insert_before(head, item, d)
7159
7160 elseif crep and crep.space then
7161   -- 655360 = 10 pt = 10 * 65536 sp
7162   d = node.new(12, 13) -- (glue, spaceskip)
7163   local quad = font.getfont(item_base.font).size or 655360
7164   node.setglue(d, crep.space[1] * quad,
7165                  crep.space[2] * quad,
7166                  crep.space[3] * quad)
7167   if mode == 0 then
7168     placeholder = ' '  
7169   end
7170   head, new = node.insert_before(head, item, d)
7171
7172 elseif crep and crep.spacefactor then
7173   d = node.new(12, 13) -- (glue, spaceskip)
7174   local base_font = font.getfont(item_base.font)
7175   node.setglue(d,
7176                  crep.spacefactor[1] * base_font.parameters['space'],
7177                  crep.spacefactor[2] * base_font.parameters['space_stretch'],
7178                  crep.spacefactor[3] * base_font.parameters['space_shrink'])
7179   if mode == 0 then
7180     placeholder = ' '  
7181   end
7182   head, new = node.insert_before(head, item, d)
7183
7184 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7185   -- ERROR
7186
7187 elseif crep and crep.penalty then
7188   d = node.new(14, 0) -- (penalty, userpenalty)
7189   d.attr = item_base.attr
7190   d.penalty = crep.penalty
7191   head, new = node.insert_before(head, item, d)
7192
7193 elseif crep and crep.space then
7194   -- Shared by disc, space and penalty.
7195   if sc == 1 then
7196     word_head = head
7197 end
7198 if crep.insert then

143
7197        w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7198        table.insert(w_nodes, sc, new)
7199    else
   7200       w_nodes[sc] = d
   7201      node.remove(head, item)
   7202      w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
   7203    end
   7204
   7205    last_match = utf8.offset(w, sc+1+step)
   7206::next::
   7207
   7208   end -- for each replacement
   7209
   7210   if Babel.debug then
   7211      print('.....', '/
   7212      Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
   7213    end
   7214
   7215   end -- for match
   7216::next::
   7217   end -- for patterns
   7218::next::
   7219   word_head = nw
   7220    end -- for substring
   7221
   7222 return head
   7223end

   7224-- This table stores capture maps, numbered consecutively
   7225Babel.capture_maps = {}
   7226
   7227-- The following functions belong to the next macro
   7228function Babel.capture_func(key, cap)
   7229    local ret = "[[. .. cap:gsub('([0-9])', ', "])(your) .. ([" "] .. "]"
   7230    local cnt
   7231    local u = unicode.utf8
   7232    ret, cnt = ret:gsub('([[0-9]])|([" ]+)]|([" ]+))', Babel.capture_func_map)
   7233    if cnt == 0 then
   7234      ret = u.gsub(ret, '{%x%x%x%x+}',
   7235      function (n)
   7236        return u.char(tonumber(n, 16))
   7237      end)
   7238    end
   7239    ret = ret:gsub("[%[\%]%\%.", ",")
   7240    ret = ret:gsub("%.%.\[\%%\]%", ",")
   7241    return key .. [[\[\%(function(m) return \[\)] . ret . [[ \[ end]
   7242end

   7243function Babel.capt_map(from, mapno)
   7244    return Babel.capture_maps[mapno][from] or from
   7245end

   7246-- Handle the {n|abc|ABC} syntax in captures
   7247function Babel.capture_func_map(capno, from, to)
   7248    local u = unicode.utf8
   7249    from = u.gsub(from, '{%x%x%x%x+}',
   7250      function (n)
   7251        return u.char(tonumber(n, 16))
   7252      end)
   7253    to = u.gsub(to, '{%x%x%x%x+}',
   7254      function (n)
   7255end

144
7260       return u.char(tonumber(n, 16))
7261   end)
7262 local froms = {}
7263 for s in string.utfcharacters(from) do
7264   table.insert(froms, s)
7265 end
7266 local cnt = 1
7267 table.insert(Babel.capture_maps, {})
7268 local mlen = table.getn(Babel.capture_maps)
7269 for s in string.utfcharacters(to) do
7270   Babel.capture_maps[mlen][froms[cnt]] = s
7271   cnt = cnt + 1
7272 end
7273 return "]]..Babel.capt_map(m[" .. capno .. "],"..
7274   (mlen) .. ")" .. "]["
7275 end
7276
7277-- Create/Extend reversed sorted list of kashida weights:
7278function Babel.capture_kashida(key, wt)
7279   wt = tonumber(wt)
7280   if Babel.kashida_wts then
7281     for p, q in ipairs(Babel.kashida_wts) do
7282       if wt == q then
7283         break
7284       elseif wt > q then
7285         table.insert(Babel.kashida_wts, p, wt)
7286         break
7287       elseif table.getn(Babel.kashida_wts) == p then
7288         table.insert(Babel.kashida_wts, wt)
7289       end
7290     end
7291   else
7292     Babel.kashida_wts = { wt }
7293   end
7294   return 'kashida = ' .. wt
7295 end
7296
7297-- Experimental: applies prehyphenation transforms to a string (letters
7298-- and spaces).
7299function Babel.string_prehyphenation(str, locale)
7300   local n, head, last, res
7301   head = node.new(8, 0) -- dummy (hack just to start)
7302   last = head
7303   for s in string.utfvalues(str) do
7304     if s == 20 then
7305       n = node.new(12, 0)
7306     else
7307       n = node.new(29, 0)
7308       n.char = s
7309     end
7310     node.set_attribute(n, Babel.attr_locale, locale)
7311     last.next = n
7312     last = n
7313   end
7314   head = Babel.hyphenate_replace(head, 0)
7315   res = ''
7316   for n in node.traverse(head) do
7317     if n.id == 12 then
7318       res = res .. ' '
7319     elseif n.id == 29 then
7320       res = res .. unicode.utf8.char(n.char)
7321     end
7322   end
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:

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

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

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

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

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

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

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

```lua
Babel = Babel or {}
Babel.bidi_enabled = true
require('babel-data-bidi.lua')
local characters = Babel.characters
local ranges = Babel.ranges
local DIR = node.id("dir")
local function dir_mark(head, from, to, outer)
  dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
  local d = node.new(DIR)
  d.dir = '+' .. dir
  node.insert_before(head, from, d)
  d = node.new(DIR)
  d.dir = '-' .. dir
  node.insert_after(head, to, d)
end
```
```lua
function Babel.bidi(head, ispar)
    local first_n, last_n -- first and last char with nums
    local last_es -- an auxiliary 'last' used with nums
    local first_d, last_d -- first and last char in L/R block
    local dir, dir_real

    Next also depends on script/lang (<al>/<r>). To be set by babel. tex.pardir is dangerous, could be
    (re)set but it should be changed only in vmode. There are two strong's – strong = l/al/r and
    strong_lr = l/r (there must be a better way):

    local strong = ('TRT' == tex.pardir) and 'r' or 'l'
    local strong_lr = (strong == 'l') and 'l' or 'r'
    local outer = strong

    local new_dir = false
    local first_dir = false
    local inmath = false
    local last_lr

    local type_n = ''

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

            Next is based on the assumption babel sets the language AND switches the script with its dir. We
            treat a language block as a separate Unicode sequence. The following piece of code is executed at the
            first glyph after a 'dir' node. We don't know the current language until then. This is not exactly true,
            as the math mode may insert explicit dirs in the node list, so, for the moment there is a hack by brute
            force (just above).

            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
                    ..
```

147
Numbers. The dual `<al>` system for R is somewhat cumbersome.

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 strong == `<al>` then
  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.

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_marks(head, first_n, last_n, `'`)
    elseif strong_lr == `<l` and first_d and type_n == `<an` then
      dir_marks(head, first_n, last_n, `'`)
    else
      dir_marks(head, first_d, last_d, outer)
    end
  end
  type_n = `'`
end

R text in L, or L text in R. Order of dir_marks's are relevant: d goes outside n, and therefore it's emitted after. See dir_marks 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, whatsitx, etc., are ignored:

```plaintext
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`, resp, 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.

```plaintext
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).

```plaintext
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
end
```

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

```plaintext
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.

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

```plaintext
⟨/basic-r⟩
```
And here the Lua code for bidi=basic:

```lua
7496 ⟨*basic⟩
7497 Babel = Babel or {}
7498
7499 -- eg, Babel.fontmap[1][<prefontid>]==<dirfontid>
7500
7501 Babel.fontmap = Babel.fontmap or {}
7502 Babel.fontmap[0] = {} -- l
7503 Babel.fontmap[1] = {} -- r
7505
7506 Babel.bidi_enabled = true
7507 Babel.mirroring_enabled = true
7508
7509 require('babel-data-bidi.lua')
7510
7511 local characters = Babel.characters
7512 local ranges = Babel.ranges
7513
7514 local DIR = node.id('dir')
7515 local GLYPH = node.id('glyph')
7516
7517 local function insert_implicit(head, state, outer)
7518   local new_state = state
7519   if state.sim and state.eim and state.sim ~= state.eim then
7520     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
7521     local d = node.new(DIR)
7522     d.dir = '+' .. dir
7523     node.insert_before(head, state.sim, d)
7524     local d = node.new(DIR)
7525     d.dir = '-' .. dir
7526     node.insert_after(head, state.eim, d)
7527   end
7528   new_state.sim, new_state.eim = nil, nil
7529   return head, new_state
7530 end
7531
7532 local function insert_numeric(head, state)
7533   local new
7534   local new_state = state
7535   if state.san and state.ean and state.san ~= state.ean then
7536     local d = node.new(DIR)
7537     d.dir = '+TLT'
7538     new, new = node.insert_before(head, state.san, d)
7539     if state.san == state.sim then state.sim = new end
7540     local d = node.new(DIR)
7541     d.dir = '-TLT'
7542     new, new = node.insert_after(head, state.ean, d)
7543     if state.ean == state.eim then state.eim = new end
7544   end
7545   new_state.san, new_state.ean = nil, nil
7546   return head, new_state
7547 end
7548
7549 -- TODO - \hbox with an explicit dir can lead to wrong results
7550 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7551 -- was made to improve the situation, but the problem is the 3-dir
7552 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7553 -- well.
7554
7555 function Babel.bidi(head, ispar, hdir)
7556   local d -- d is used mainly for computations in a loop
7557   local prev_d = ''
7558```
local new_d = false
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
-- when the callback is called, we are just after the box,
-- and the textdir is that of the surrounding text
-- if not ispar and hdir ~= tex.textdir then
-- 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 item.id == GLYPH
        or (item.id == 7 and item.subtype == 2) then
        local d_font = nil
        local item_r
        if item.id == 7 and item.subtype == 2 then
            item_r = item.replace -- automatic discs have just 1 glyph
        else
            item_r = item
        end
        local chardata = characters[item_r.char]
        d = chardata and chardata.d or nil
        if not d or d == 'nsm' then
            for nn, et in ipairs(ranges) do
                if item_r.char < et[1] then
                    break
                elseif item_r.char <= et[2] then
                    break
                end
            end
        else
            if d == 'nsm' then
                if item_r.char == 1 then
                    break
                elseif item_r.char <= et[2] then
                    break
                end
            end
            if item_r.char == 1 then
                break
            end
        end
    end
    -- three cases: glyph, dir, otherwise
    -- if item.id == 7 and item.subtype == 2 then
    -- else
    -- end
end
if not d then d = et[3]

elseif d == 'nsm' then d_font = et[3]

break

end

end

d = d or 'l'

-- A short 'pause' in bidi for mapfont

d_font = d_font or d

d_font = (d_font == 'l' and 0) or
    (d_font == 'nsm' and 0) or
    (d_font == 'r' and 1) or
    (d_font == 'al' and 2) or
    (d_font == 'an' and 2) or nil

if d_font and fontmap and fontmap[d_font][item_r.font] then
    item_r.font = fontmap[d_font][item_r.font]
end

if new_d then
    table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})

if inmath then
    attr_d = 0
else
    attr_d = node.get_attribute(item, ATDIR)
    attr_d = attr_d & 0x3
end

if attr_d == 1 then
    outer_first = 'r'
    last = 'r'
else if attr_d == 2 then
    outer_first = 'r'
    last = 'al'
else
    outer_first = 'l'
    last = 'l'
end

outer = last
has_en = false
first_et = nil
new_d = false
end

if glue_d then
    if (d == 'l' and 'l' or 'r') ~= glue_d then
        table.insert(nodes, {glue_i, 'on', nil})
    end

    glue_d = nil
    glue_i = nil
end

elseif item.id == DIR then

d = nil

if head ~= item then new_d = true end

elseif item.id == node.id'glue' and item.subtype == 13 then
    glue_d = d
    glue_i = item
    d = nil
else if item.id == node.id'math' then

152
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
    end
else
    temp = 'on'
    -- W6
end
for e = first_et, #nodes do
    if nodes[e][1].id == GLYPH 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
end

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

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

local first_on = nil
for q = 1, #nodes do
    local item
    local outer_first = nodes[q][3]
    outer = outer_first or outer
    last = outer_first or last
    local d = nodes[q][2]
    if d == 'an' or d == 'en' then d = 'r' end
    if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
    if d == 'on' then
        first_on = first_on or q
    elseif first_on then
        if last == d then
            temp = d
        else
            temp = outer
        end
        for r = first_on, q - 1 do
            nodes[r][2] = temp
            item = nodes[r][1] -- MIRRORING
            if Babel.mirroring_enabled and item.id == GLYPH

154
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
end
if d == 'r' or d == 'l' then last = d end
end
-- IMPLICIT, REORDER ----------------
outer = save_outer
last = outer
local state = {}
state.has_r = false
for q = 1, #nodes do
  local item = nodes[q][1]
  outer = nodes[q][3] or outer
  local d = nodes[q][2]
  if d == 'nsm' then d = last end -- W1
  if d == 'en' then d = 'an' end
  local isdir = (d == 'r' or d == 'l')
  if outer == 'l' and d == 'an' then
    state.san = state.san or item
    state.ean = item
    elseif state.san then
      head, state = insert_numeric(head, state)
    end
    if outer == 'l' then
      if d == 'an' or d == 'r' then -- im -> implicit
        if d == 'r' then state.has_r = true end
        state.sim = state.sim or item
        elseif d == 'l' and state.sim and state.has_r then
          head, state = insert_implicit(head, state, outer)
        elseif d == 'l' then
          state.sim, state.eim, state.has_r = nil, nil, false
        end
        else
          if d == 'an' or d == 'l' then
            if nodes[q][3] then -- nil except after an explicit dir
              state.sim = item -- so we move sim 'inside' the group
            else
              state.sim = state.sim or item
            end
            state.eim = item
            elseif d == 'r' and state.sim then
              head, state = insert_implicit(head, state, outer)
            end
          end
      end
    end
  end
end
  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
end
if d == 'r' or d == 'l' then last = d end
end
end
-- IMPLICIT, REORDER ----------------
outer = save_outer
last = outer
local state = {}
state.has_r = false
for q = 1, #nodes do
  local item = nodes[q][1]
  outer = nodes[q][3] or outer
  local d = nodes[q][2]
  if d == 'nsm' then d = last end -- W1
  if d == 'en' then d = 'an' end
  local isdir = (d == 'r' or d == 'l')
  if outer == 'l' and d == 'an' then
    state.san = state.san or item
    state.ean = item
    elseif state.san then
      head, state = insert_numeric(head, state)
    end
    if outer == 'l' then
      if d == 'an' or d == 'r' then -- im -> implicit
        if d == 'r' then state.has_r = true end
        state.sim = state.sim or item
        elseif d == 'l' and state.sim and state.has_r then
          head, state = insert_implicit(head, state, outer)
        elseif d == 'l' then
          state.sim, state.eim, state.has_r = nil, nil, false
        end
        else
          if d == 'an' or d == 'l' then
            if nodes[q][3] then -- nil except after an explicit dir
              state.sim = item -- so we move sim 'inside' the group
            else
              state.sim = state.sim or item
            end
            state.eim = item
            elseif d == 'r' and state.sim then
              head, state = insert_implicit(head, state, outer)
            end
          end
      end
    end
  end
end
155
elseif d == 'r' then
  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
        linking = 0
      end
    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 pattern 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}[⟨⟨date⟩⟩ v⟨⟨version⟩⟩] Nil language
\LdfInit{nil}{datenil}

When this file is read as an option, i.e. by the \usepackage command, nil could be an ‘unknown’ language in which case we have to make it known.

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

This macro is used to store the values of the hyphenation parameters \lefthyphenmin and \righthyphenmin.
\providehyphenmins{\CurrentOption}{\m@ne\m@ne}

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

\captionnil\datenil
\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}{Latn}%
\bbl@elt{identification}{script.tag.opentype}{DFLT}%
\bbl@elt{identification}{level}{1}%
\bbl@elt{identification}{encodings}{}}% Derivate
\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@sotf@nil}{latn}

The macro \ldf@finish takes care of looking for a configuration file, setting the main language to be switched on at \begin{document} and resetting the category code of @ to its original value.

\ldf@finish{nil}
\ldf{nil}

\caption{nil}
\datenil
13 Calendars

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

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

\[ \text{Compute Julian day} \equiv \text{\texttt{bbl@fpmod(#1,#2)floor(#1/#2)}} \]
\[ \text{bbl@cs@gregleap(#1)} \&\& ( (!((\text{\texttt{bbl@fpmod(#1,100)}} == 0) \&\& (\text{\texttt{bbl@fpmod(#1,400)}} != 0))) ) \]
\[ \text{\texttt{bbl@cs@jd(year, month, day)}} \]
\[ \text{\texttt{bbl@ca@islamic-civil++}} \text{\texttt{bbl@ca@islamic-civil+}} \text{\texttt{bbl@ca@islamic-civil}} \text{\texttt{bbl@ca@islamic-civil-}} \text{\texttt{bbl@ca@islamic-civil--}} \]

13.1 Islamic

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

\[ \text{\texttt{bbl@ca@islamic-civil}} \text{\texttt{bbl@ca@islamic-civil-}} \]

The Umm al-Qura calendar, used mainly in Saudi Arabia, is based on moment-hijri, by Abdullah Alsiger (license MIT).

Since the main aim is to provide a suitable \today, and maybe some close dates, data just covers Hijri ~1455/1460 (Gregorian ~2014/2038).
This is basically the set of macros written by Michail Rozman in 1991, with corrections and adaptations by Rama Porrat, Misha, Dan Haran and Boris Lavva. This must be eventually replaced by computations with l3fp. An explanation of what's going on can be found in hebcal.sty
#4=\bbl@cntcommon}
\newif\ifbbl@hebrleap
\def\bbl@checkleaphebryear#1{%\countdef\tmpa=0\countdef\tmpb=1\tmpa=#1\relax\multiply\tmpa by 7\advance\tmpa by 1\bbl@remainder{\tmpa}{19}{\tmpb}\ifnum\tmpb<7\global\bbl@hebrleaptrue\else\global\bbl@hebrleapfalse\fi}}\def\bbl@hebrelapsedmonths#1#2{%\countdef\tmpa=0\countdef\tmpb=1\countdef\tmpc=2\tmpa=#1\relax\advance\tmpa by -1#2=\tmpa\divide\tmpa by 19\divide#2 by 19\multiply#2 by 235\bbl@remainder{\tmpa}{19}{\tmpb}%\tmpa=years%19-years this cycle\tmpc=\tmpb\multiply\tmpb by 12\advance\tmpb by \tmpa\multiply\tmpc by 7\advance\tmpc by 1\divide\tmpc by 19\advance\tmpc by 2\global\bbl@cntcommon=\#2\global\#2=\bbl@cntcommon}\def\bbl@hebrelapseddays#1#2{%\countdef\tmpa=0\countdef\tmpb=1\countdef\tmpc=2\bbl@hebrelapsedmonths{#1}{#2}\tmpa=#2\relax\multiply\tmpa by 13753\advance\tmpa by 5604\bbl@remainder{\tmpa}{25920}{\tmpc}%\tmpc == ConjunctionParts\divide\tmpa by 25920\multiply#2 by 29\advance#2 by 1\advance#2 by \tmpa\bbl@remainder{#2}{7}{\tmpa}\ifnum\tmpc<19440\ifnum\tmpc<9924\else\ifnum\tmpc<16789\else\ifnum\tmpc<19440\else\ifnum\tmpc<9924\else\ifnum\tmpc=2\bbl@checkleaphebryear{#1}% of a common year\ifbbl@hebrleap\else\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi

161
There is an algorithm written in TeX by Jabri, Abolhassani, Pournader and Esfahbod, created for the first versions of the FarsiTeX system (no longer available), but the original license is GPL, so its use with LPPL is problematic. The code here follows loosely that by John Walker, which is free and accurate, but sadly very complex, so the relevant data for the years 2013-2050 have been
pre-calculated and stored. Actually, all we need is the first day (either March 20 or March 21).

\ExplSyntaxOn
\def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
\def\bbl@ca@persian#1-#2-#3\@@#4#5#6{% 
\edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
\ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
\bbl@afterfi\expandafter\@gobble
\fi\fi
\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#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
\edef#5{\fp_eval:n{(\#6 <= 186) ? ceil((\#6 - 6) / 30) : ceil((\#6 - 6) / 30))}}
\ExplSyntaxOff
}\ExplSyntaxOff
\ExplSyntaxOn
\ExplSyntaxOff

13.4 Coptic and Ethiopic

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

\ExplSyntaxOn
\ExplSyntaxOff
\ExplSyntaxOn
13.5 Buddhist

That's very simple.

That's very simple.

\begin{macrocode}
\ExplSyntaxOn
\def\bbl@ca@chinese#1-#2-#3\@@#4#5#6{\def\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{885,1920,2953,3809,4873,5906,6881,7825,8898,9893,10778}\def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,354,384,413,443,472,501,531,560,590,620,649,679,709,738,768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,2923,2953,2982,3011,3041,3071,3101,3130,3160,3189,3219,3248,3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,4341,4370,4400,4430,4460,4490,4518,4547,4577,4606,4635,4665,4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,2015-2044.}}\begin{macrocode}
\ExplSyntaxOff
\end{macrocode}
\end{document}
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 local\_hyphen\_tex\ or whatever they like, but they mustn’t diddle with \hyphen\\_\tex\ (or plain\_\tex\ except to preload additional fonts).

The files \bplain\\_\tex\ and \blplain\\_\tex\ can be used as replacement wrappers around plain\_\tex\ and lplain\_\tex\ to achieve the desired effect, based on the babel package. If you load each of them with \ini\TeX, 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 ini\TeX\ sees, we need to set some category codes just to be able to change the definition of \input.

\begin{verbatim}
\catcode`{=1 % left brace is begin-group character
\catcode`}=2 % right brace is end-group character
\catcode`#=6 % hash mark is macro parameter character
\end{verbatim}

If a file called \hyphen\_cfg can be found, we make sure that it will be read instead of the file \hyphen\\_\tex\. We do this by first saving the original meaning of \input and I use a one letter control sequence for that so as not to waste multi-letter control sequence on this in the format).

\begin{verbatim}
\openin \hyphen\_cfg
\ifeof\hyphen\_cfg
\let\input\undefined
\fi
\end{verbatim}

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

\begin{verbatim}
\def\input #1 {\let\input\undefined}
\end{verbatim}

\ExplSyntaxOff

\langle /ca-chinese \rangle
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 bplain.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\2ε style file. So, in Plain we must provide at least some predefined values as well some tools to set them (even if not all options are available). There are no package options, and therefore and alternative mechanism is provided. For the moment, only \babeloptionstrings and \babeloptionmath are provided, which can be defined before loading babel. \BabelModifiers can be set too (but not sure it works).

```latex
\def\@empty{}
\def\loadlocalcfg#1{\openin0#1.cfg\ifeof0\closein0\else\closein0\immediate\write16{************************************}\immediate\write16{* Local config file #1.cfg used}\immediate\write16{*}\input #1.cfg\relax\fi}\@endofldf
```

### 14.3 General tools

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

```latex
\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\long#1}}\let\l@ngrel@x\relax\def\@car#1#2\@nil{#1}\def\@cdr#1#2\@nil{#2}\let\@typeset@protect\relax\protected@edef\edef\long\def\@gobble#1{}\edef\@backslashchar{\expandafter\@gobble\string\}\def\g@addto@macro#1#2{\toks\expandafter{#1#2}\xdef#1{\the\toks}}\def\@namedef#1{\expandafter\def\csname #1\endcsname}\def\@nameuse#1{\csname #1\endcsname}
```

167
\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#2{\begingroup\newlinechar=`\^^J\def\{^^J(babel)\}%\errhelp{#2}\errmessage{\#1}\endgroup}
\def\bbl@warning#1{\begingroup\newlinechar=`\^^J\def\{^^J\}%\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}}
\@onlypreamble\@onlypreamble
\def\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.
\def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
\@onlypreamble\AtEndOfPackage
\let\bbl@afterlang\@empty
\chardef\bbl@opt@hyphenmap\z@
\LaTeX needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default. There is a trick to hide some conditional commands from the outer `\ifx`. The same trick is applied below.

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

Mimic \LaTeX's commands to define control sequences.

\begin{verbatim}
\def\newcommand\@star@or@long\new@command{
\def\new@command#1{\@testopt\@newcommand#1\relax\one}
\def\@newcommand#1[#2]{\@ifnextchar\[\@xargdef#1[#2]}{\@argdef#1[#2]}
\def\@argdef#1[#2]{\@yargdef#1\relax\@ne{#2}}
\def\@xargdef#1[#2]\[#3]{\expandafter\def\expandafter#1\expandafter{\@protected@testopt\csname\string#1\endcsname\tw@{#2}{#3}}}
\def\@yargdef#1#2#3{\@tempcnta#3\relax\advance\@tempcnta\one\let\@hash@\relax\edef\reserved@a{\ifx#2\tw@\[\@hash@1\]\fi}\@tempcntb#2\@whilenum\@tempcntb<\@tempcnta\do{\edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}}\let\@hash@\@\longrelax\expandafter\def\expandafter#1\reserved@a}
\def\providecommand\@star@or@long\provide@command{
\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}
\def\DeclareRobustCommand\@star@or@long\declare@robustcommand{
\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\expandafter\csname\expandafter\@gobble\string#1\endcsname}{\noexpand\protect\expandafter\@gobble\csname\expandafter\@gobble\string#1\endcsname}}
\def\De\endgroup
\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\expandafter\csname\expandafter\@gobble\string#1\endcsname}{\noexpand\protect\expandafter\@gobble\csname\expandafter\@gobble\string#1\endcsname}
\end{verbatim}

169
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}##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 \@ifl@aded checks whether a file was loaded. This functionality is not needed for plain \TeX but we need the macro to be defined as a no-op.

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

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

\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
\let\reserved@d=#1%
\def\reserved@a{#2}\def\reserved@b{#3}\
\futurelet\@let@token\@ifnch
\def\@ifnch{\ifx\@let@token\@sptoken\let\reserved@c\@xifnch\else
\ifx\@let@token\reserved@d\let\reserved@c\reserved@a\else\let\reserved@c\reserved@b\fi\fi}
\else
\let\reserved@a\reserved@d
\def\reserved@d{#2}\def\reserved@b{#3}\
\futurelet\@let@token\@ifnch
\def\@ifnch{\ifx\@let@token\@sptoken\let\reserved@c\reserved@d\else
\ifx\@let@token\reserved@b\let\reserved@c\reserved@b\else\let\reserved@c\reserved@a\fi\fi}
14.4 Encoding related macros

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

\begin{verbatim}
\def\DeclareTextCommand{\@dec@text@cmd\providecommand}
\def\ProvideTextCommand{\@dec@text@cmd\providecommand}
\def\DeclareTextSymbol#1#2#3{\@dec@text@cmd\chardef#1{#2}#3}
\def\@dec@text@cmd#1#2#3{\expandafter\def\expandafter#2\expandafter{\csname#3-cmd\expandafter\endcsname#2\csname#3\string#2\endcsname}}
\def\@current@cmd#1{\ifx\protect\@typeset@protect\else\noexpand#1\expandafter\@gobble\fi}
\def\@changed@cmd#1#2{\ifx\protect\@typeset@protect\expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax\expandafter\ifx\csname ?\string#1\endcsname\relax\expandafter\def\csname ?\string#1\endcsname{\@changed@x@err{#1}}\else\global\expandafter\let\csname\cf@encoding\string#1\endcsname\csname ?\string#1\endcsname\fi\fi\else\noexpand#1\fi}
\def\@current@cmd#1{\ifx\protect\@typeset@protect\else\noexpand#1\expandafter\@gobble\fi}
\end{verbatim}
Currently we only use the E\TeX{}\texttt{2\epsilon} method for accents for those that are known to be made active in some language definition file.

\begin{verbatim}
\def\UseTextSymbol#1#2{#2}
\def\UseTextAccent#1#2#3{}
\def\@use@text@encoding#1{}
\def\DeclareTextSymbolDefault#1#2{\DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}}
\def\DeclareTextAccentDefault#1#2{\DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}}
\cf@encoding{OT1}
\end{verbatim}

The following control sequences are used in babel.def but are not defined for plain \TeX{}.

\begin{verbatim}
\let\scriptsize\sevenrm
\end{verbatim}

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

\begin{verbatim}
\ifx\scriptsize@undefined\let\scriptsize\sevenrm\fi
\end{verbatim}

And a few more “dummy” definitions.

\begin{verbatim}
\let\bbl@opt@shorthands\@nnil
\let\bbl@ifshorthand#1#2#3{#2}
\let\bbl@language@opts\@empty
\let\bbl@ensureinfo\@gobble
\let\bbl@provide@locale\relax
\let\bbl@opt@strings\@undefined
\else\let\bbl@opt@strings\babeloptionstrings\fi
\def\BabelStringsDefault{generic}
\def\bbl@tempa{normal}
\ifx\babeloptionmath\bbl@tempa\def\bbl@mathnormal{\noexpand\textormath}\fi
\def\AfterBabelLanguage#1#2{}
\ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
\let\bbl@afterlang\relax
\def\bbl@opt@safe{BR}
\ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
\ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
\expandafter\newif\csname ifbbl@single\endcsname
\chardef\bbl@bidimode\z@
\langle/Emulate LaTeX\rangle
\end{verbatim}

A proxy file:
15 Acknowledgements

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

References