Babel

Localization and internationalization

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

1 Identification and loading of required files

*Code documentation is still under revision.*

The babel package after unpacking consists of the following files:

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

There 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 (e.g., 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 (e.g., there are no geographic areas in Spanish). Not all include LICR variants.

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

See *Keys in ini files* in the the babel site.

3 Tools

```
1 ⟨⟨version=3.96⟩⟩
2 ⟨⟨date=2023/10/25⟩⟩
```

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

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

```
3 ⟨⟨Basic macros⟩⟩ ≡
4 \bbl@trace{Basic macros}
5 \def\bbl@stripslash{\expandafter\@gobble\string}
6 \def\bbl@add#1#2{%
7 \bbl@ifunset{\bbl@stripslash#1}{%
8 {\def#1{#2}}%
9 {\expandafter\def\expandafter#1\expandafter{#2}}}
10 \def\bbl@xin@{\@expandtwoargs\in@}
11 \def\bbl@carg#1#2{\expandafter#1\csname#2\endcsname}%
12 \def\bbl@ncarg#1#2#3{\expandafter#1\expandafter#2\csname#3\endcsname}%
13 \def\bbl@ccarg#1#2#3{\expandafter#1\csname#2\expandafter\endcsname\csname#3\endcsname}%
14 \def\bbl@csarg#1#2{\expandafter\csname bbl@#2\endcsname}%
15 \def\bbl@cl#1{\csname bbl@#1@\languagename\endcsname}%
16 \def\bbl@carg#1#2#3{\expandafter#1\expandafter#2\csname#3\endcsname}%
17 \def\bbl@cl#1{\csname#1\bbl@#1@\languagename\endcsname}%
```
\def\bbl@loop#1#2#3{\bbl@@loop#1{#3}#2,\@nnil,}
\def\bbl@loopx#1#2{\expandafter\bbl@loop\expandafter#1\expandafter{#2}}
\def\bbl@@loop#1#2#3,{\ifx\@nnil#3\relax\else\def#1{#3}#2\bbl@afterfi\bbl@@loop#1{#2}%%%%%%%%%%%%%%%%%%%
\fi}
\def\bbl@for#1#2#3{\bbl@loopx#1{#2}{\ifx#1\@empty\else#3\fi}}
\bbl@add@list
This internal macro adds its second argument to a comma separated list in its first argument. When the list is not defined yet (or empty), it will be initiated. It presumes expandable character strings.
\def\bbl@add@list#1#2{\edef#1{\bbl@ifunset{\bbl@stripslash#1}{}{\ifx#1\@empty\else#1,\fi}#2}}
\bbl@afterelse
Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the \else and \fi parts of an if-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...\fi statement appears in one of the arguments and it is not enclosed in braces.
\long\def\bbl@afterelse#1\else#2\fi{\fi#1}
\long\def\bbl@afterfi#1\fi{\fi#1}
\bbl@exp
Now, just syntactical sugar, but it makes partial expansion of some code a lot more simple and readable. Here \texttt{\\ldots} stands for \texttt{\noexpand\ldots} for \noexpand applied to a built macro name (which does not define the macro if undefined to \relax, because it is created locally), and \texttt{\[\ldots\]} for one-level expansion (where \ldots 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\@nil\relax\relax#3{#3{#1}}\long\def\bbl@trim@def#1{\bbl@trim{\def#1}}
\bbl@ifunset
To check if a macro is defined, we create a new macro, which does the same as \@ifundefined. However, in an \epsilon-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
\edef\bbl@ifunset#1{%\expandafter\ifx\csname#1\endcsname\relax
\else
 \expandafter\@firstoftwo
\fi}\bbl@ifunset{ifcsname}{}{\edef\bbl@ifunset#1{%\ifcsname#1\endcsname\expandafter\ifx\csname#1\endcsname\relax
\bbl@afterelse\expandafter\@firstoftwo\else
 \bbl@afterfi\expandafter\@secondoftwo\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{#4}\def\bbl@ifset#1#2#3{%\bbl@ifunset{#1}{#3}{\bbl@exp{\bbl@ifblank{\@nameuse{#1}}}{#3}{#2}}}

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

\def\bbl@forkv#1#2{%\def\bbl@kvcmd##1##2##3{#2}\bbl@kvnext#1,\@nil,}\def\bbl@kvnext#1,{%\ifx\@nil#1\relax\else\bbl@ifblank{#1}{}{\bbl@forkv@eq#1=#2=#3\@nil#4}\expandafter\bbl@kvnext\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 \texttt{for} loop. Each item (trimmed), is \texttt{#1}. It cannot be nested (it's doable, but we don't need it).

\def\bbl@foreach#1{%\expandafter\bbl@vforeach\expandafter{#1}}\def\bbl@vforeach#1#2{%\def\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}\def\bbl@forcmd{#1}\{\bbl@trim\bbl@vforeach{\bbl@forcmd{#1}}{\@nil}{\@nil}{#2}{#4}}

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

\def\bbl@replace#1#2#3{% in \texttt{#1} -> repl \texttt{#2} by \texttt{#3}\toks@{}\def\bbl@replace@aux#1#2#3#4{%\expandafter\bbl@replace@aux#1#2#3#4}}
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 `\bbl@TG@@date`, and also fails if there are macros with spaces, because they are retokenized). It may change! (or even merged with `\bbl@replace`; I’m not sure checking the replacement is really necessary or just paranoia).

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

3.1 Multiple languages

Plain \TeX version 3.0 provides the primitive \language that is used to store the current language. When used with a pre-3.0 version this function has to be implemented by allocating a counter. The following block is used in switch.def and hyphen.cfg; the latter may seem redundant, but remember babel doesn't require loading switch.def in the format.
Another counter is used to keep track of the allocated languages. \texttt{\LaTeX} and \texttt{\MiKTeX} reserves for this purpose the count 19.

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

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

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

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

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

Formal declaration:

```latex
\NeedsTeXFormat{LaTeX2e}[2005/12/01]
\ProvidesPackage{babel}[\langle\langle date\rangle\rangle v\langle\langle version\rangle\rangle The Babel package]
```


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@infowarn\@gobble
\let\bbl@warning\@gobble
\afterbabellanguage#1{% 
\global\expandafter\bbl@add\csname#1.ldf-h@@k\endcsname}

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

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

### 3.3 base

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

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

```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
```

9
3.4 key=value options and other general option

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

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

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

\DeclareOption{KeepShorthandsActive}{}
\DeclareOption{activeacute}{}
\DeclareOption{activegrave}{}
\DeclareOption{debug}{}
\DeclareOption{noconfigs}{}
\DeclareOption{showlanguages}{}
\DeclareOption{silent}{}
% \DeclareOption{mono}{}
\DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
% \chardef\bbl@iniflag\z@% main -> +1
% \chardef\bbl@iniflag\twe@% add = 2
% \chardef\bbl@iniflag\thr@@% add + main
% A separate option
% \newif\bbl@single
% \DeclareOption{selectors=off}{\bbl@singletrue}
% \DeclareOption{shorthandsoff}{\bbl@tempa shorthandsoff=\bbl@tempa}
% \chardef\bbl@iniflag\z@% main -> +1
% \DeclareOption{provide=*}{\chardef\bbl@iniflag\one}
% \DeclareOption{provide==*}{\chardef\bbl@iniflag\tw@}% add = 2
% \DeclareOption{provide===*}{\chardef\bbl@iniflag\thr@@}% add + main
% Don't use. Experimental. TODO.
% \newif\bbl@single
% \DeclareOption{selectors=off}{\bbl@singletrue}
% \DeclareOption{(More package options)}

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

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

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

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

\let\bbl@language@opts@\empty
\DeclareOption*{%
  \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*

\ifx\bbl@opt@provide@\nnil
  \let\bbl@opt@provide@\empty % %\% MOVE above
\else
  \chardef\bbl@iniflag@\one
  \bbl@exp{\bbl@forkv\{\nameuse{\raw@opt@babel.sty}\}\{%\CurrentOption}}%
    \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=....

\def\bbl@sh@string#1{%
  \ifx#1\empty\else
    \ifx#1t\string~%
    \else\ifx#1c\string,%
    \else\string#1%
The following macro tests if a shorthand is one of the allowed ones.

\def\bbl@ifshorthand#1#2#3{#2}%
\ifx\bbl@opt@shorthands\@nnil
\def\bbl@ifshorthand#1{%
\bbl@xin{"\string#1}{\bbl@opt@shorthands}%
\ifin@
\expandafter\@firstoftwo
\else
\expandafter\@secondoftwo
\fi}
\fi

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

\def\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{%\set@typeset@protect\select@language@x\expandafter{\bbl@opt@headfoot}\\let\protect\noexpand\fi

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

\g@addto@macro\@resetactivechars{%\let\protect\noexpand\fi

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*.

\expandafter@secondoftwo
\fi
\fi
⟨/package⟩

⟨∗package⟩

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*.

```latex
\expandafter\@secondoftwo
\fi
\fi
⟨/package⟩

⟨∗package⟩
```

4 Multiple languages

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

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

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

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

```latex
\def\adddialect#1#2{%
  \global\chardef#1#2\relax
  \bbl@usehooks{adddialect}{(#1)(#2)}%
  \begingroup
  \count@#1\relax
  \def\bbl@elt##1##2##3##4{%
    \ifnum\count@=##2\relax
      \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}\
      \bbl@info{Hyphen rules for \expandafter\@gobble\bbl@tempa set to \string\csname l@##1\endcsname\%
        (\string\language\the\count@). Reported}\
    \fi}\
  \bbl@cs{languages}\
  \endgroup}
```

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

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

```latex
\def\bbl@fixname#1{%
  \begingroup
  \def\bbl@tempe{l@}\
  \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}\bbl@info{Hyphen rules for \expandafter\@gobble\bbl@tempa set to \string\csname l@##1\endcsname\%
    (\string\language\the\count@). Reported}\
  \bbl@cs{languages}\
  \endgroup}
```

13
After a name has been ‘fixed’, the selectors will try to load the language. If even the fixed name is not defined, we will load it on the fly, either based on its name, or if activated, its BCP 47 code.

We first need a couple of macros for a simple BCP lookup. It also makes sure, with \bbl@bcp{...}, casing is the correct one, so that sr-latn-ba becomes fr-Latin-BA. Note #4 may contain some \@empty’s, but they are eventually removed. \bbl@bcplookup either returns the found ini or it is \relax.

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

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

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

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

\bbl@pop@language

But when the language change happens inside a group the end of the group doesn't write anything to the auxiliary files. Therefore we need \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@language@stack

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

\bbl@push@language
\bbl@pop@language

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

\bbl@push@language
\bbl@pop@language

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@pop@lang

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@pop@lang

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

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

\localeid

% No real need for a new counter
\chardef\localeid@z@
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 \\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) 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 for the macro definition of \originalTeX, we construct the control sequence name for the \noextras\langle lang\rangle command at definition time by expanding the \csname primitive.

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

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

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

\newif\ifbbl@usedategroup
\let\bbl@savedextras\@empty
\def\bbl@switch#1{% from select@, foreign@
% make sure there is info for the language if so requested
\ensureinfo[\#1]\%\restore
% originalTeX
\expandafter\def\expandafter\originalTeX\expandafter{\%
\csname noextras\langle lang\rangle\endcsname
\let\originalTeX\@empty
\bbl@beginsave}%
\bbl@usehooks{afterreset}%
% set the locale id
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 \extrastlang command doesn’t make any \global changes. The coding is very similar to part of \selectlanguage.

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

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

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

\providecommand\bbl@beforeforeign{}
\edef\foreignlanguage{\protect\expandafter\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}\let\bbl@select@opts=#1\BabelText{#2}{\par}\bbl@usehooks{foreign}{}\BabelText{#3}}}
\def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
\begingroup{\par}\def\bbl@selectorname{foreign*}\let\bbl@select@opts\@empty\BabelText{#1}{\par}\bbl@dirparastext\BabelText{#2}}}
\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}}}
\def\IfBabelSelectorTF#1{\bbl@xin{,\bbl@selectorname,}{,\zap@space#1 \@empty,}\ifin@\expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
\bbl@patterns This macro selects the hyphenation patterns by changing the \language register. If special hyphenation patterns are available specifically for the current font encoding, use them instead of the default.
It also sets hyphenation exceptions, but only once, because they are global (here language \lccode's has been set, too). \bbl@hyphenation@ is set to relax until the very first \babelhyphenation, so do nothing with this value. If the exceptions for a language (by its number, not its name, so that :ENC is
taken into account) has been set, then use \hyphenation with both global and language exceptions and empty the latter to mark they must not be set again.

\begin{verbatim}
\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=\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}{\bbl@tempa}\%
% > luatex\@ifundefined{bbl@hyphenation@}{\begingroup\bbl@xin@,\number\language,}{,\bbl@hyphlist}\ifin@\else\@expandtwoargs\bbl@usehooks{hyphenation}{\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\}
\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}\fi\expandafter\expandafter\expandafter\set@hyphenmins\csname\bbl@tempf hyphenmins\endcsname\relax\fi\bbl@usehooks{hyphenation}{\bbl@tempf}\hyphenation{\@empty\bbl@hyphenation%@\space\csname bbl@hyphenation%@\endcsname}}\bbl@fixname\bbl@tempf\set@hyphenmins{\bbl@tempf hyphenmins\endcsname}\relax\set@hyphenmins{\bbl@tempf hyphenmins\endcsname}\relax\set@hyphenmins{\bbl@tempf hyphenmins\endcsname}\relax\bbl@fixname\bbl@tempf\endgroup}}
\end{verbatim}

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

\begin{verbatim}
\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}\fi\bbl@usehooks{hyphenation}{\bbl@tempf}\hyphenation{\@empty\bbl@hyphenation%@\space\csname bbl@hyphenation%@\endcsname}}\bbl@fixname\bbl@tempf\set@hyphenmins{\bbl@tempf hyphenmins\endcsname}\relax\set@hyphenmins{\bbl@tempf hyphenmins\endcsname}\relax\set@hyphenmins{\bbl@tempf hyphenmins\endcsname}\relax\bbl@fixname\bbl@tempf\endgroup}
\end{verbatim}

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

\begin{verbatim}
\def\providehyphenmins#1#2{%\expandafter\ifx\csname #1hyphenmins\endcsname\relax\@namedef{#1hyphenmins}[#2]\fi}
\end{verbatim}

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

\begin{verbatim}
\def\sethyphenmins#1#2{%....
\end{verbatim}
The identification code for each file is something that was introduced in \TeX\2ε. When the \LaTeX\ command \ProvidesFile does not exist, a dummy definition is provided temporarily. For use in the language definition file the command \ProvidesLanguage is defined by babel. Depending on the format, ie, on if the former is defined, we use a similar definition or not.

\ProvidesLanguage

\iffalse
\ProvidesFile\@undefined
\else
\ProvidesLanguage#1{%
 \wlog{Language: #1 #4 #3 <#2>}%
}
\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.
\if\originalTeX\@undefined\let\originalTeX\@empty\fi

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

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.

\noolpterr

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.
\addto\protect\bbl@tempa{#1}\n\bbl@sreplace\bbl@tempa{\languagename}\n\bbl@warning{\@backslashchar#1 not set for '\languagename'. Please,\ndefine it after the language has been loaded\n(typically in the preamble) with:\n\string\setlocalecaption{\languagename}{\bbl@tempa}{..}\nFeel free to contribute on github.com/latex3/babel.\nReported}}
\def\bbl@tentative{\protect\bbl@tentative@i}\n\def\bbl@tentative@i#1{\n\bbl@warning{Some functions for '#1' are tentative.\nThey might not work as expected and their behavior\ncould change in the future.\nReported}}
\def\@nolanerr#1{\bbl@error{You haven't defined the language '#1' yet.\nPerhaps you misspelled it or your installation\nis not complete}\n{Your command will be ignored, type <return> to proceed}}
\def\@nopatterns#1{\bbl@warning{No hyphenation patterns were preloaded for\nthe language '#1' into the format.\nPlease, configure your TeX system to add them and\nrebuild the format. Now I will use the patterns\npreloaded for \bbl@nulllanguage\space instead}}
\let\bbl@usehooks\@gobbletwo\n\ifx\bbl@onlyswitch\@empty\endinput\fi\n% Here ended switch.def
\@nolanerr{language}\n\@nopatterns{language}\n\ifx\directlua\@undefined\else\n\ifx\bbl@luapatterns\@undefined\input luababel.def\fi\n\fi\n\bbl@trace{Compatibility with language.def}\n\ifx\bbl@languages\@undefined\else\input language.def\fi\n\chardef\l@english\z@\n\fi\n\addto\protect\bbl@tempa{\l@english}\n\def\addlanguag#1#2#3#4#5{%\n\expandafter\ifx\csname lang@#1\endcsname\relax\else\n\global\expandafter\let\csname l@#1\expandafter\endcsname\n\csname lang@#1\endcsname\n\fi}\n\def\uselanguag#1{%\n\input language.def\n\else\n\message{I couldn't find the file language.def}\n\else\closein\n\message{I couldn't find the file language.def}\n\endgroup\n\def\addlanguage#1#2#3#4#5{%\n\expandafter\ifx\csname lang@#1\endcsname\relax\else\n\global\expandafter\let\csname l@#1\expandafter\endcsname\n\csname lang@#1\endcsname\n\fi}\n\def\uselanguag#1{%\n\input language.def\n\endgroup
\fi\n\chardef\l@english\z@\n\fi
\addto\protect\bbl@tempa{\l@english}\n\addto It takes two arguments, a \emph{control sequence} and \TeX-code to be added to the \emph{control sequence}.
If the ⟨control sequence⟩ has not been defined before it is defined now. The control sequence could also expand to ⟨\relax⟩, in which case a circular definition results. The net result is a stack overflow. Note there is an inconsistency, because the assignment in the last branch is global.

\def\addto#1#2{\ifx#1\@undefined\def#1{#2}\else\ifx#1\relax\def#1{#2}\else\toks@\expandafter{#1#2}\xdef#1{\the\toks@}\fi\fi}

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

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

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

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

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

\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\langle\bbl@tempa\space}}}\bbl@exp{\let\langle\org@\bbl@tempa\langle\bbl@tempa\space}}}\@namedef{\bbl@tempa\space}}

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

\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{\langle\bbl@tempa\space}}}\bbl@exp{\let\langle\org@\bbl@tempa\langle\bbl@tempa\space}}}\@namedef{\bbl@tempa\space}}

4.3 Hooks

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

\def\bbl@usehooks{\bbl@trace{Hooks}
\newcommand\AddBabelHook[3][]{}\bbl@ifunset{\bbl@hk@#2}{\EnableBabelHook[#2]}}%
\def\bbl@tempa##1,#3=##2,##3\@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}\[
\bbl@tempb\]}
\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 (?)\n\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
\fi}
To ensure forward compatibility, arguments in hooks are set implicitly. So, if a further argument is added in the future, there is no need to change the existing code. Note events intended for hyphen.cfg are also loaded (just in case you need them for some reason).
\def\bbl@evargs{,% <- don't delete this comma
everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
beforestart=0,languagename=2,begindocument=1}
\ifx\NewHook\@undefined\else % Test for Plain (?)\n\def\bbl@tempa#1=#2\@@{\NewHook{babel/#1}}
\bbl@foreach\bbl@evargs{\bbl@tempa#1\@@}\fi
\babelensure
The user command just parses the optional argument and creates a new macro named \bbl@e@⟨language⟩. We register a hook at the afterextras event which just executes this macro in a "complete" selection (which, if undefined, is \relax and does nothing). This part is somewhat involved because we have to make sure things are expanded the correct number of times. The macro \bbl@e@⟨language⟩ contains \bbl@ensure{(include)}{(exclude)}{(fontenc)}, which in turn loops over the macros names in \bbl@captionslist, excluding (with the help of \in@) those in the exclude list. If the fontenc is given (and not \relax), the fontencoding is also added. Then we loop over the include list, but if the macro already contains \foreignlanguage, nothing is done. Note this macro (1) is not restricted to the preamble, and (2) changes are local.
4.4 Setting up language files

\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 \ldfquit 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`.

\bbl@trace{Macros for setting language files up}
\def\bbl@ldfinit{%
  \let\bbl@screset\@empty
  \let\BabelStrings\bbl@opt@string
  \let\BabelOptions\@empty
  \let\BabelLanguages\relax
  \ifx\originalTeX\@undefined
    \let\originalTeX\@empty
  \else
    \originalTeX
  \fi}
\def\LdfInit#1#2{%
  \chardef\atcatcode=\catcode`@\relax
  \catcode`@=11\relax
  \chardef\eqcatcode=\catcode`=\relax
  \catcode`==12\relax
  \expandafter\if\expandafter\@backslashchar
    \expandafter\@car\string#2\@nil
    \ifx#2\@undefined
      \ldf@quit{#1}%
    \else
      \ldf@quit{#1}%
    \fi
  \else
    \expandafter\ifx\csname#2\endcsname\relax
      \ldf@quit{#1}%
    \fi
  \fi
  \bbl@ldfinit}
\ldf@quit This macro interrupts the processing of a language definition file.
\def\ldf@quit#1{%
  \expandafter\main@language\expandafter{#1}%
  \catcode`@=\atcatcode \let\atcatcode\relax
  \catcode`==\eqcatcode \let\eqcatcode\relax
}\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.
\def\bbl@afterldf#1{%
  \bbl@afterlang
  \let\BabelModifiers\relax
  \let\bbl@screset\relax
}%
\def\ldf@finish#1{%
  \loadlocalcfg{#1}%
  \bbl@afterldf{#1}%
  \expandafter\main@language\expandafter{#1}%
  \catcode`@=\atcatcode \let\atcatcode\relax
  \catcode`==\eqcatcode \let\eqcatcode\relax
}\endinput

After the preamble of the document the commands `\LdfInit`, `\ldf@quit` and `\ldf@finish` are no longer needed. Therefore they are turned into warning messages in \LaTeX.
This command should be used in the various language definition files. It stores its argument in \bbl@main@language; to be used to switch to the correct language at the beginning of the document.

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

We also have to make sure that some code gets executed at the beginning of the document, either when the 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.

\begin{verbatim}
1157 \def\bbl@beforestart{% 
1158 \def\@nolanerr##1{% 
1159 \bbl@warning{Undefined language '##1' in aux.\Reported}}% 
1160 \bbl@usehooks{beforestart}{}% 
1161 \global\let\bbl@beforestart\relax}
1162 \AtBeginDocument{% 
1163 {\@nameuse{bbl@beforestart}}% Group! 
1164 \if@filesw 
1165 \providecommand\babel@aux[2]{}% 
1166 \immediate\write\@mainaux{\string\providecommand\string\babel@aux[2]{}}% 
1167 \immediate\write\@mainaux{\@nameuse{bbl@beforestart}}% 
1168 \fi 
1169 \expandafter\selectlanguage\expandafter{\bbl@main@language}% 
1170 \if\bbl@single % must go after the line above. 
1171 \renewcommand\selectlanguage[1]{}% 
1172 \renewcommand\foreignlanguage[2]{#2}% 
1173 \global\let\babel@aux\gobbletwo % Also as flag 
1174 \fi 
1175 \if\bbl@single % must go after the line above. 
1176 \renewcommand\selectlanguage[1]{}% 
1177 \renewcommand\foreignlanguage[2]{#2}% 
1178 \fi 
1179 \global\let\babel@aux\gobbletwo % Also as flag 
1180 \fi 
1181 \if\bbl@single % must go after the line above. 
1182 \renewcommand\selectlanguage[1]{}% 
1183 \renewcommand\foreignlanguage[2]{#2}% 
1184 \fi 
1185 \AddToHook{begindocument/before}{% 
1186 \let\bbl@normalsf=\normalsf % Hack, to delay the setting 
1187 \ifcase\bbl@engine\or 
1188 \AtBeginDocument{\pagedir\bodydir} % TODO - a better place 
1189 \fi 
1190 \AddToHook{begindocument/before}{% 
1191 \let\bbl@normalsf=\normalsf % Hack, to delay the setting 
1192 \ifcase\bbl@engine\or 
1193 \AtBeginDocument{\pagedir\bodydir} % TODO - a better place 
1194 \fi 
\end{verbatim}

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

\begin{verbatim}
1195 \def\select@language[#1]{% 
1196 \ifcase\bbl@select@type 
1197 \bbl@ifsamestring\languagename[#1]{\selectlanguage[#1]}% 
1198 \else 
1199 \selectlanguage[#1]% 
1200 \fi 
\end{verbatim}

\subsection{Shorthands}

The macro \bbl@add@special is used to add a new character (or single character control sequence) to the macro \dospecials (and \@sanitize if \LaTeX{} is used). It is used only at one place, namely
when \initiate@active@char is called (which is ignored if the char has been made active before). Because \@sanitize can be undefined, we put the definition inside a conditional.
Items are added to the lists without checking its existence or the original catcode. It does not hurt, but should be fixed. It's already done with \nfss@catcodes, added in 3.10.

\bbl@trace{Shorthands}
\def\bbl@add@special#1{%
\bbl@add\dospecials{\do#1}% test \@sanitize = \relax, for back. compat.
\bbl@ifunset{@sanitize}{}{\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{\@makeother#1}%
\else
\endgroup\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.

\bbl@active@def#1#2#3#4{%
@@namedef{#3#1}{%
\expandafter\ifx\csname#2@sh@#1@\endcsname\relax
\bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
\else
\bbl@afterfi\csname#2@sh@#1@\endcsname
\fi}%
\begin{thebibliography}{9}
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 doublequote character active one could have \initiate@active@char{"} in a language definition file. This defines " as \active@prefix "\active@char" (where the first " is the character with its original catcode, when the shorthand is created, and \active@char" is a single token). In protected contexts, it expands to \protect " or \noexpand " (ie, with the original "); otherwise \active@char" is executed. This macro in turn expands to \normal@char" in “safe” contexts (eg, \label), but \user@active" in normal “unsafe” ones. The latter search a definition in the user, language and system levels, in this order, but if none is found, \normal@char" is used. However, a deactivated shorthand (with \bbl@deactivate is defined as \active@prefix "\normal@char". The following macro is used to define shorthands in the three levels. It takes 4 arguments: the (string'ed) character, ⟨level⟩@group, ⟨level⟩@active and ⟨next-level⟩@active (except in system).

\begin{verbatim}
\def\bbl@active@def#1#2#3#4{%\@namedef{#3#1}{%\expandafter\ifx\csname#2@sh#1\endcsname\relax\bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%;\else\bbl@afterfi\csname#2@sh#1\endcsname
\fi}%
\end{verbatim}

When there is also no current-level shorthand with an argument we will check whether there is a next-level defined shorthand for this active character.
\long@namedef{#3@arg#1}{{\%\\expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax\\bbl@afterelse\csname#4#1\endcsname##1\\%\\else\\bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname\\fi}}\\initiate@active@char calls \initiate@active@char with 3 arguments. All of them are the same character with different catcodes: active, other (\string'ed) and the original one. This trick simplifies the code a lot.

\def\initiate@active@char#1{\\bbl@ifunset{active@char\string#1}{}{\\bbl@withactive{\\expandafter\@initiate@active@char\expandafter}#1\string#1#1}}\\initiate@active@char calls \initiate@active@char with 3 arguments. All of them are the same character with different catcodes: active, other (\string'ed) and the original one. This trick simplifies the code a lot.

The very first thing to do is saving the original catcode and the original definition, even if not active, which is possible (undefined characters require a special treatment to avoid making them \relax and preserving some degree of protection).

\def\@initiate@active@char#1#2#3{\\bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}\\ifx#1\@undefined\\bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}\\else\\bbl@csarg\let{oridef@@#2}#1\bbl@csarg\edef{oridef@#2}{\\let\noexpand#1\\expandafter\noexpand\csname bbl@oridef@@#2\endcsname}\\fi}If the character is already active we provide the default expansion under this shorthand mechanism. Otherwise we write a message in the transcript file, and define \normal@char{⟨char⟩} to expand to the character in its default state. If the character is mathematically active when babel is loaded (for example ‘’) the normal expansion is somewhat different to avoid an infinite loop (but it does not prevent the loop if the mathcode is set to “8000 a posteriori”).

\ifx#1#3\relax\\expandafter\let\csname normal@char#2\endcsname normal@char{⟨char⟩}to expand to the character in its default state. If the character is mathematically active when babel is loaded (for example ‘’) the normal expansion is somewhat different to avoid an infinite loop (but it does not prevent the loop if the mathcode is set to “8000 a posteriori”).

To prevent problems with the loading of other packages after babel we reset the catcode of the character to the original one at the end of the package and of each language file (except with \text{KeepShorthandsActive}). It is re-activate again at \text{\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 \text{\cite} for example. Then we make it active (not strictly necessary, but done for backward compatibility).

\bbl@restoreactive{#2}\AtBeginDocument{\catcode`#2\active\\if\files\\immediate\write\@mainaux{\catcode`\string#2\active}\\fi\\expandafter\bbl@add@special\csname#2\endcsname\\catcode`#2\active\\fi}Now we have set \normal@char{⟨char⟩}, we must define \active@char{⟨char⟩}, to be executed when the character is activated. We define the first level expansion of \active@char{⟨char⟩} to check the
status of the @safe@actives flag. If it is set to true we expand to the ‘normal’ version of this character; otherwise we call \user@active(char) to start the search of a definition in the user, language and system levels (or eventually normal@char(char)).

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

Wenowdefine the default values which the shorthand is set to when activated or deactivated. It is set to the deactivated form (globally), so that the character expands to 
\active@prefix (char) \normal@char(char)
(where \active@char(char) is one control sequence!).

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

Thenextlevel 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\csname normal@char#2\endcsname}%
\expandafter\edef\csname user@group @sh@#2@\string\protect\endcsname{\expandafter\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\pr@m@s\bbl@pr@m@s
\let\active@math@prime\@percent
\fi
\bbl@usehooks{initiateactive}{(#1)(#2)(#3)}
The following package options control the behavior of shorthands in math mode.

\begin{verbatim}
1314 ⟨⟨More package options⟩⟩ ≡
1315 \DeclareOption{math=active}{\let\bbl@restoreactive\@gobble}
1316 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1317 ⟨⟨/More package options⟩⟩

Initiating a shorthand makes active the char. That is not strictly necessary but it is still done for backward compatibility. So we need to restore the original catcode at the end of package and the end of the ldf.

\begin{verbatim}
1318 \iffpackagewith{babel}\{KeepShorthandsActive\}
1319 \let\bbl@restoreactive@gobble
1320 \\AfterBabelLanguage\CurrentOption
1321 \\AtEndOfPackage
1322 \\catcode`#1=\the\catcode`#1\relax
1323 \\AtEndOfPackage{\catcode`#1=\the\catcode`#1\relax}
\end{verbatim}

\bbl@sh@select

This command helps the shorthand supporting macros to select how to proceed. Note that this macro needs to be expandable as do all the shorthand macros in order for them to work in expansion-only environments such as the argument of \hyphenation. This macro expects the name of a group of shorthands in its first argument and a shorthand character in its second argument. It will expand to either \bbl@firstcs or \bbl@scndcs. Hence two more arguments need to follow it.

\begin{verbatim}
1327 \def\bbl@sh@select#1#2{\expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1328 \bbl@afterelse\bbl@scndcs
1329 \bbl@afterelse\bbl@firstcs
1330 \else
1331 \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1332 \fi}
\end{verbatim}

\active@prefix

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

\begin{verbatim}
1333 \edef\active@prefix#1{%
1334 \if\csname#1\endcsname\relax
1335 \if\protect\@typeset@protect
1336 \else
1337 \else
1338 \protect\@unexpandable@protect
1339 \noexpand\protect\string#1\relax
1340 \else
1341 \protect\#1
1342 \fi
1343 \\expandafter\@gobble
1344 \fi}
1345 \edef\active@prefix#1{%
1346 \if\csname#1\endcsname
1347 \string#1\relax
1348 \\expandafter\\expandafter\expandafter\@gobble
1349 \else
1350 \if\protect\@typeset@protect
1351 \else
1352 \if\protect\@unexpandable@protect
1353 \noexpand\protect\string#1\relax
1354 \else
1355 \protect\#1
1356 \fi
1357 \\expandafter\expandafter\expandafter\@gobble
1358 \fi
\end{verbatim}

33
In some circumstances it is necessary to be able to reset the shorthand to its ‘normal’ value (usually the character with catcode ‘other’) on the fly. For this purpose the switch `\@safe@actives` is available. The setting of this switch should be checked in the first level expansion of `\active@char`\langle char\rangle. When this expansion mode is active (with `\@safe@actives`true), something like \char{13}_{13} becomes \char{12}_{12} in an `\edef` (in other words, shorthand 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\langle char\rangle` in the case of `\@activate`, or `\normal@char\langle char\rangle` in the case of `\@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@texpdf` improves the interoperativity with hyperref and takes 4 arguments: (1) The \TeX{} code in text mode, (2) the string for hyperref, (3) the \TeX{} code in math mode, and (4), which is currently ignored, but it’s meant for a string in math mode, like a minus sign instead of an hyphen (currently hyperref doesn’t discriminate the mode). This macro may be used in \lf{} files.

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@texpdf` improves the interoperativity with hyperref and takes 4 arguments: (1) The \TeX{} code in text mode, (2) the string for hyperref, (3) the \TeX{} code in math mode, and (4), which is currently ignored, but it’s meant for a string in math mode, like a minus sign instead of an hyphen (currently hyperref doesn’t discriminate the mode). This macro may be used in \lf{} files.
Some of the shorthands that will be declared by the language definition files have to be usable in both text and math mode. To achieve this the helper macro \textormath is provided.

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

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

\def\useshorthands{%
  \ifstar\bbl@usesh@s{\bbl@usesh@x{}}
  \def\bbl@usesh@s#1{%
    \bbl@usesh@x
    \bbl@ifshorthand{#1}{%
      \def\user@group{user}
      \initiate@active@char{#1}
      #1
    }{\bbl@error
      \{I can't declare a shorthand turned off (\string#1)\}
      \{Sorry, but you can't use shorthands which have been\%
      turned off in the package options\}}}

\def\defineshorthand{%
  \bbl@ifshorthand{#1}%
  \bbl@ifshorthand{#2}{%
    \\def\user@group{user}%
    \\initiate@active@char{#2}%
    #2
  }{\bbl@error
    \{I can't declare a shorthand turned off (\string#2)\}
    \{Sorry, but you can't use shorthands which have been\%
\languageshorthands  A user level command to change the language from which shorthands are used. Unfortunately, babel currently does not keep track of defined groups, and therefore there is no way to catch a possible change in casing to fix it in the same way languages names are fixed. [TODO].

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

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

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

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

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

\shorthandoff  The macro \bbl@switch@sh takes the list of characters apart one by one and subsequently switches the category code of the shorthand character according to the first argument of \bbl@switch@sh. But before any of this switching takes place we make sure that the character we are dealing with is known as a shorthand character. If it is, a macro such as \active@char" should exist.
Switching off and on is easy – we just set the category code to ‘other’ (12) and \active. With the starred version, the original category code and the original definition, saved in \initiate\active\char, are restored.

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

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

\def\babelshorthand{(\active@prefix\babelshorthand\bbl@putsh) \% 
\def\bbl@putsh#1{\% 
  \bbl@ifunset{bbl@active@\string#1}\{\% 
    \csname bbl@active@\string#1\endcsname \% 
  \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 \% 
  \def\initiate@active@\char\#1\{\% 
    \bbl@s@initiate@active@\char{\#1}\} \% 
  \let\bbl@s@switch@sh@\bbl@switch@sh\% 
  \def\bbl@s@switch@sh@\#1\#2\% 
  \ifx\#2\@nnil\else \% 
    \bbl@afterfi \% 
  \fi \% 
  \let\bbl@s@activate@\bbl@activate \% 
  \def\bbl@s@deactivate@\bbl@deactivate \% 
\fi} \% 

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

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

Usually the ~ is active and expands to \penalty\@M\hspace{1pt}. When it is written to the .aux file it is written
expanded. To prevent that and to be able to use the character ~ as a start character for a shorthand,
it is redefined here as a one character shorthand on system level. The system declaration is in most
cases redundant (when ~ is still a non-break space), and in some cases is inconvenient (if ~ has been
redefined); however, for backward compatibility it is maintained (some existing documents may rely
on the babel value).

\begin{verbatim}
\begin{group}
catcode`^=7 \catcode`*=\active \lccode`*=`
catcode`\"=12 \catcode`\"=`
\lowercase{\gdef\bbl@pr@m@s{\bbl@if@primes\''\pr@@@s\{\bbl@if@primes\*\pr@@@t\egroup}}}
\endgroup
\end{verbatim}

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.

\begin{verbatim}
bbl@trace{Language attributes}
\newcommand\languageattribute[2]{\def\bbl@tempc[#1]\%
bbl@fixname\bbl@tempc\bbl@iflanguage\bbl@tempc\%
\bbl@foreach[2]{\bbl@vforeach}}
\end{verbatim}
We want to make sure that each attribute is selected only once; therefore 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.

```
1571 \ifx\bbl@known@attribs\undefined
1572 \in@false
1573 \else
1574 \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
1575 \fi
1576 \ifin@
1577 \bbl@warning{%
1578 You have more than once selected the attribute ‘##1’\%
1579 for language #1. Reported}%
1580 \else
1581 When we end up here the attribute is not selected before. So, we add it to the list of selected attributes and execute the associated \TeX-code.
1582 \bbl@exp{%
1583 \bbl@add@list\bbl@known@attribs{\bbl@tempc-##1}}%
1584 \edef\bbl@tempa{\bbl@tempc-##1}%
1585 \expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes%
1586 {\csname\bbl@tempc @attr@##1\endcsname}%
1587 \else}}}%
1588 \@onlypreamble\languageattribute
1589 \newcommand*{\@attrerr}[2]{%
1590 \bbl@error
1591 {The attribute #2 is unknown for language #1.}%
1592 {Your command will be ignored, type <return> to proceed}}%
```

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

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

\bbl@ifattributeset This internal macro has 4 arguments. It can be used to interpret \TeX-code based on whether a certain attribute was set. This command should appear inside the argument to \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.

```
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}
```

\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.

```
1611 \def\bbl@ifknown@trib#1\#2{%
1612 \let\bbl@tempa@secondoftwo
1613 \bbl@loopx\bbl@tempb{#2}{%\11\expandafter\in\expandafter{\expandafter,,\bbl@tempb,}{,#1,}%
1614 \ifin@\11\let\bbl@tempa@firstoftwo\else\fi}%
1615 \bbl@tempa}
```

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

```
1620 \def\bbl@clear@ttribs{%
1621 \ifx\bbl@attributes\@undefined\else
1622 \bbl@loopx\bbl@tempa{\bbl@attributes}{%\expandafter\bbl@clear@ttrib\bbl@tempa.}%
1623 \let\bbl@attributes\@undefined
1624 \fi}
```

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 \originalTeX). Note undefined macros are not undefined anymore when saved – they are \texttt{\relax}'d.

```
1629 \def\babel@save#1{%
1630 \def\bbl@tempa{{,#1,}}% Clumsy, for Plain
1631 \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%\expandafter\bbl@savedextras,}%
1632 \expandafter\in\bbl@tempa\ifin@\else\bbl@add\bbl@savedextras{,#1,}%
1633 \bbl@carg\let{babel\number\babel@savecnt}#1\relax
1634 \toks@\expandafter{\originalTeX\let#1=}%
1635 \bbl@exp{\texttt{\originalTeX}\let\the\toks@<\bbl@number\bbl@savecnt>\relax}%
```

\texttt{\originalTeX} has to be expandable, i.e. you shouldn't let it to \texttt{\relax}.  

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

```
1629 \bbl@trace{Macros for saving definitions}
1630 \def\ babel@beginsave{\babel@savecnt\z@}
```

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

```
1631 \newcount\bbl@savecnt
1632 \bbl@beginsave
```

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

```
1633 \def\babel@save#1{%
1634 \def\bbl@tempa{\#,1,}\texttt{Clumsy, for Plain}
1635 \bbl@exp{\texttt{\originalTeX}\let\the\toks@<\bbl@number\bbl@savecnt>{1,}%
1636 \bbl@exp{\texttt{\originalTeX}\let\the\toks@<\bbl@number\bbl@savecnt>{1,}%
1637 \bbl@exp{\texttt{\originalTeX}\let\the\toks@<\bbl@number\bbl@savecnt>}{1,}%
1638 \bbl@exp{\texttt{\originalTeX}\let\the\toks@<\bbl@number\bbl@savecnt>}{1,}%
1639 \bbl@exp{\texttt{\originalTeX}\let\the\toks@<\bbl@number\bbl@savecnt>}{1,}%
1640 \bbl@exp{\texttt{\originalTeX}\let\the\toks@<\bbl@number\bbl@savecnt>}{1,}%
1641 \bbl@exp{\texttt{\originalTeX}\let\the\toks@<\bbl@number\bbl@savecnt>}{1,}%
1642 \bbl@exp{\texttt{\originalTeX}\let\the\toks@<\bbl@number\bbl@savecnt>}{1,}%
1643 \bbl@exp{\texttt{\originalTeX}\let\the\toks@<\bbl@number\bbl@savecnt>}{1,}%
1644 \bbl@exp{\texttt{\originalTeX}\let\the\toks@<\bbl@number\bbl@savecnt>}{1,}%
```

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

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

4.8 Short tags

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

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

\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.

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.

\nobreak \hskip 0pt plus 0pt\relax.
There should not be a discretionary after a hyphen at the beginning of a word, so it is prevented if preceded by a skip. Unfortunately, this does handle cases like “(-suffix)”. \nobreak is always preceded by \leavevmode, in case the shorthand starts a paragraph.

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

The following macro inserts the hyphen char.

1742 \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 ldf’s. After a space, the \mbox in \bbl@hy@nobreak is redundant.

1749 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}} 1750 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}} 1751 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1753 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1754 \def\bbl@hy@nobreak{\bbl@usehyphen\mbox{\bbl@hyphenchar}}
1755 \def\bbl@hy@nobreak{\bbl@usehyphen\mbox{\bbl@hyphenchar}}
1757 \def\bbl@hy@repeat{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1758 \def\bbl@hy@repeat{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1760 \def\bbl@hy@empty{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1762 \def\bbl@hy@empty{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}

\bbl@disc For some languages the macro \bbl@disc is used to ease the insertion of discretionaries for letters that behave ‘abnormally’ at a breakpoint.
1763 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{\bbl@hyphenchar}{#1}\bbl@allowhyphens}

4.10 Multiencoding strings

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

Tools But first, a tool. It makes global a local variable. This is not the best solution, but it works.
1764 \bbl@trace{Multiencoding strings}
1765 \def\bbl@togo@local#1{\global\let#1\@empty}

The second one. We need to patch \@uclclist, but it is done once and only if \SetCase is used or if strings are encoded. The code is far from satisfactory for several reasons, including the fact \@uclclist is not a list any more. Therefore a package option is added to ignore it. Instead of gobbling the macro getting the next two elements (usually \reserved@a), we pass it as argument to \bbl@uc1c. The parser is restarted inside \langle\lang\rangle@bbl@uc1c because we do not know how many expansions are necessary (depends on whether strings are encoded). The last part is tricky – when uppercasing, we have:

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

and starts over (and similarly when lowercasing).
1766 \@ifpackagewith{babel}{nocase}{%
1767 \let\bbl@patchuclc\relax%
The following package options control the behavior of \SetString.

\DeclareOption{nocase}{}
\DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
\DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
\def\BabelStringsDefault{generic}

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

\newcommand{\bbl@startcmds@ii}[1][\@empty]{%
There are two versions of \bbl@scswitch. The first version is used when ldfs are read, and it makes sure \langle group\rangle \langle language\rangle is reset, but only once (\bbl@screset is used to keep track of this). The second version is used in the preamble and packages loaded after babel and does nothing. The macro \bbl@forlang loops \bbl@L but its body is executed only if the value is in \BabelLanguages (inside babel) or \date\langle language\rangle is defined (after babel has been loaded). There are also two version of \bbl@forlang. The first one skips the current iteration if the language is not in \BabelLanguages (used in ldfs), and the second one skips undefined languages (after babel has been loaded).

Now we define command to be used inside \StartBabelCommands.

**Strings**  The following macro is the actual definition of \SetString when it is “active” First save the “switcher”. Create it if undefined. Strings are defined only if undefined (ie, like \providescommand). With the event stringprocess you can preprocess the string by manipulating the value of \BabelString. If there are several hooks assigned to this event, preprocessing is done in the same order as defined. Finally, the string is set.
Now, some additional stuff to be used when encoded strings are used. Captions then include \bbl@encoded for string to be expanded in case transformations. It is \relax by default, but in \MakeUppercase and \MakeLowercase its value is a modified expandable \@changed@cmd.

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

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

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

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.
Macros local to BabelCommands

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

\newcommand\BabelLower[2]{\ifnum\lccode#1=#2\else\babel@savevariable{\lccode#1}\lccode#1=#2\relax\fi}
\newcommand\BabelLowerMM[4]{\@tempcnta=#1\relax\@tempcntb=#4\relax\def\bbl@tempa{\ifnum\@tempcnta>#2\else\@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}\advance\@tempcnta#3\relax\advance\@tempcntb#3\relax\expandafter\bbl@tempa\fi}\bbl@tempa}
\newcommand\BabelLowerMO[4]{\@tempcnta=#1\relax\def\bbl@tempa{\ifnum\@tempcnta>#2\else\@expandtwoargs\BabelLower{\the\@tempcnta}{#4}\advance\@tempcnta#3\expandafter\bbl@tempa\fi}\bbl@tempa}

The following package options control the behavior of hyphenation mapping.

\DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\@undefined}
\DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
\DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
\DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
\DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}

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\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}#1\fi\bbl@tempa}

48
4.11 Macros common to a number of languages
\setlowbox
The following macro is used to lower quotes to the same level as the comma. It prepares its argument in box register 0.
\setlowbox#1{\settowhitespace{\box#1}{\box#1}{\box#1}}
\save@sf@q
The macro \save@sf@q is used to save and reset the current space factor.
\save@sf@q#1{\leavevmode\begin{enumerate}}
\efdef@SF{\spacefactor\the\spacefactor\#1}
4.12.1 Quotation marks

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

\begin{verbatim}
2071 \ProvideTextCommand{\quotedblbase}{OT1}{%
2072 \save@sf@q{\set@low@box{\textquotedblright}/}%
2073 \box\z@\kern-.04em\bbl@allowhyphens}}
2074 \ProvideTextCommandDefault{\quotedblbase}{%
2075 \UseTextSymbol{OT1}{\quotedblbase}}
\end{verbatim}

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

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

\begin{verbatim}
2076 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2077 \save@sf@q{\set@low@box{\textquotationright}/}%
2078 \box\z@\kern-.04em\bbl@allowhyphens}}
2079 \ProvideTextCommandDefault{\quotesinglbase}{%
2080 \UseTextSymbol{OT1}{\quotesinglbase}}
\end{verbatim}

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

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

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

Make sure that when an encoding other than OT1 or T1 is used these glyphs can still be typeset.
The single guillemets are not available in OT1 encoding. They are faked.

\providecommand{\guilsingleleft}{OT1}{% 
  \ifmmode < \else \save@sf@q{\nobreak \raise.2ex \hbox{$\scriptscriptstyle<$} \bbl@allowhyphens} \fi}

\providecommand{\guilsingleright}{OT1}{% 
  \ifmmode > \else \save@sf@q{\nobreak \raise.2ex \hbox{$\scriptscriptstyle>$} \bbl@allowhyphens} \fi}

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

\providecommanddefault{\guilsingleleft}{% 
  \usesymbol{OT1}{\guilsingleleft}}

\providecommanddefault{\guilsingleright}{% 
  \usesymbol{OT1}{\guilsingleright}}

\section*{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.

\providecommanddefault{\ij}{% 
  \usesymbol{OT1}{\ij}}

\providecommanddefault{\IJ}{% 
  \usesymbol{OT1}{\IJ}}

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

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

\def\crrtic@{\hrule height0.1ex width0.3em}

\def\crttic@{\hrule height0.1ex width0.33em}

\def\ddj@{\setbox0\hbox{d}\dimen@=.6\ht0 \advance\dimen@1ex \dimen@.45\dimen@
  \leavevmode\rlap{\raise\dimen@{\kern\dimen@\vbox{\crrtic@}}}}

\def\DDJ@{\setbox0\hbox{D}\dimen@=.55\ht0 \advance\dimen@.15\fontdimen7\font\dimen@
  \leavevmode\rlap{\raise\dimen@{\kern\dimen@\vbox{\crttic@}}}}

\def\DDJ@{% 
  \setbox0\hbox{D}\dimen@.55\ht0 \advance\dimen@.15\fontdimen7\font\dimen@
  \leavevmode\rlap{\raise\dimen@{\kern\dimen@\vbox{\crttic@}}}}

\DeclareTextCommand{\dj}{OT1}{\ddj@ D}

\DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
Make sure that when an encoding other than OT1 or T1 is used these glyphs can still be typeset.

\texttt{\ProvideTextCommandDefault{\dj}{\UseTextSymbol{OT1}{\dj}}}\par
\texttt{\ProvideTextCommandDefault{\DJ}{\UseTextSymbol{OT1}{\DJ}}}\par
\texttt{\SS}

For the T1 encoding \texttt{\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.

\texttt{\DeclareTextCommand{\SS}{OT1}{SS}}\par
\texttt{\ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}}\par

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 \texttt{\ProvideTextCommandDefault}, but this is very likely not required because their definitions are based on encoding-dependent macros.

\texttt{\glq}
\texttt{\grq}

The ‘german’ single quotes.

\texttt{\ProvideTextCommandDefault{\glq}{\textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}}\par
The definition of \texttt{\grq} depends on the font encoding. With T1 encoding no extra kerning is needed.

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

\texttt{\glqq}
\texttt{\grqq}

The ‘german’ double quotes.

\texttt{\ProvideTextCommandDefault{\glqq}{\textormath{\quotedblbase}{\mbox{\quotedblbase}}}}\par
The definition of \texttt{\grqq} depends on the font encoding. With T1 encoding no extra kerning is needed.

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

\texttt{\flq}
\texttt{\frq}

The ‘french’ single guillemets.

\texttt{\flq}
\texttt{\frq}

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.

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

\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 \langle dimen \rangle register.

\expandafter\ifx\csname U@D\endcsname\relax\csname newdimen\endcsname\U@D\fi

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

\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
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\@english\@undefined\chardef\@english\z\fi
\% The following is used to cancel rules in ini files (see Amharic).
\ifx\@unhyphenated\@undefined\newlanguage\@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}%
\bbl@trace{Input engine specific macros}

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 \TeX 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.

\begin{verbatim}
\providecommand\babelfont{\bbl@error{This macro is available only in LuaLaTeX and XeLaTeX.} Consider switching to these engines.}
\providecommand\babelprehyphenation{\bbl@error{This macro is available only in LuaLaTeX.} Consider switching to that engine.}
\ifx\babelposthyphenation\@undefined\let\babelposthyphenation\babelprehyphenation\let\babelpatterns\babelprehyphenation\let\babelcharproperty\babelprehyphenation\fi
\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.
\end{verbatim}
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
\bbl@load@basic{#2}%
% == script, language ==
% Override the values from ini or defines them
\ifx\bbl@KVP@script\@nnil\else
  \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}\
\fi
\ifx\bbl@KVP@language\@nnil\else
  \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}\
\fi
\ifcase\bbl@engine\or
  \bbl@ifunset{bbl@chrng@\languagename}{}\
  \directlua{
    Babel.set_chranges_b(\bbl@cl{sbcp}', \bbl@cl{chrng}') }
\fi
% == onchar ==
\ifx\bbl@KVP@onchar\@nnil\else
  \bbl@luahyphenate\bbl@exp{\Select@language{#2}{}}
  \directlua{
    if Babel.locale_mapped == nil then
      Babel.locale_mapped = true
      Babel.linebreaking.add_before(Babel.locale_map, 1)
    end
    Babel.chr_to_loc = Babel.loc_to_scr
  }
\fi
% == onchar ==
\ifx\bbl@KVP@onchar\@nnil\else
  \bbl@luahyphenate\bbl@exp{\Select@language{#2}{}}
  \directlua{
    if Babel.locale_props[\the\localeid].letters == false then
      Babel.locale_props[\the\localeid].letters = true
    end
  }
\fi
% == onchar ==
\ifx\bbl@KVP@onchar\@nnil\else
  \bbl@luahyphenate\bbl@exp{\Select@language{#2}{}}
  \directlua{
    if Babel.locale_props[\the\localeid].letters == true then
      Babel.locale_props[\the\localeid].letters = false
    end
  }
\fi
% == onchar ==
\ifx\bbl@KVP@onchar\@nnil\else
  \bbl@luahyphenate\bbl@exp{\Select@language{#2}{}}
  \directlua{
    if Babel.locale_props[\the\localeid].letters == false then
      Babel.locale_props[\the\localeid].letters = true
    end
  }
\fi
% == onchar ==
\ifx\bbl@KVP@onchar\@nnil\else
  \bbl@luahyphenate\bbl@exp{\Select@language{#2}{}}
  \directlua{
    if Babel.locale_props[\the\localeid].letters == true then
      Babel.locale_props[\the\localeid].letters = false
    end
  }
\fi
```
\if\bbl@xin@{ fonts } \fi
\bbl@funset{\bbl@lsys\@language}{\bbl@providelsys\@language}\% 
\bbl@funset{\bbl@wdir\@language}{\bbl@providdirs\@language}\%
\directlua{ 
\if Babel\script_blocks\[\bbl@cl{sbcp}\] then 
\text{Babel.\loc_to_scr}[\text{the\localeid}] = 
\text{Babel\script_blocks}[\bbl@cl{sbcp}] 
\end}\%
\ifx\bbl@mapselect\@undefined \fi
\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{ 
\text{Babel.locale\_props}[\text{the\csname bbl@id@@##1\endcsname}]\% 
[\bbl@prefontid'] = \fontid'\font'\space\}% 
\fi\}% 
\fi % TODO - catch non-valid values 
\fi % == mapfont == 
% For bidi texts, to switch the font based on direction 
\ifx\bbl@KVP@mapfont\@nnil\else % We can override the ini or set 
\bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}\% 
\fi 
\bbl@provide@intraspace 
\bbl@exp{% 
\bbl@add\bbl@mapselect{\\\bbl@mapdir\@language}\% 
\fi % TODO - catch non-valid values 
\fi % == Line breaking: intraspace, intrapenalty == 
\ifx\bbl@KVP@intraspace\@nnil\else % For CJK, East Asian, Southeast Asian, if interspace in ini 
\bbl@ifsamestring{\bbl@KVP@intraspace}{direction}{}\%
\bbl@error{Option '\bbl@KVP@mapfont' unknown for'}\% 
\text{mapfont. Use 'direction'.} \%
\text{See the manual for details.}}\%
\fi 
\bbl@funset{\bbl@lsys\@language}{\bbl@providelsys\@language}\% 
\bbl@funset{\bbl@wdir\@language}{\bbl@providdirs\@language}\% 
\ifx\bbl@mapselect\undefined \fi \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 % avoid font warning 
\bbl@switchfont 
\directlua{ 
\text{Babel.fontmap}[\text{the\csname bbl@wdir@@##1\endcsname}]\% 
[\bbl@prefontid'] = \fontid'\font'\space\}% 
\fi \% \bbl@exp{% 
\bbl@add\bbl@mapselect{\\\bbl@mapdir\@language}\% 
\fi 
\ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set 
\bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}\% 
\fi 
\bbl@provide@intraspace
% == Line breaking: CJK quotes == TODO -> @extras
\ifcase\bbl@engine\or
  \bbl@xin{/c}{/\bbl@cl{lnbrk}}%
\ifin@
  \bbl@ifunset{bbl@quote@\languagename}{%}
  \directlua{
    Babel.locale_props[\the\localeid].cjk_quotes = {}
    local cs = 'op'
    for c in string.utfvalues(%
      [[\csname bbl@quote@\languagename\endcsname]]) do
      if Babel.cjk_characters[c].c == 'qu' then
        Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
        cs = (cs == 'op') and 'cl' or 'op'
      end
    end
  }%
  \fi
\fi
\fi
% == Line breaking: justification ==
\ifx\bbl@KVP@justification@nil\else
  \let\bbl@KVP@linebreaking=\bbl@KVP@justification
\fi
\ifx\bbl@KVP@linebreaking@nil\else
  \bbl@xin{,\bbl@KVP@linebreaking,}%
  ,elongated,kashida,cjk,padding,unhyphenated,%
  \ifin@
    \bbl@csarg\xdef{lnbrk@\languagename}{\expandafter@car\bbl@KVP@linebreaking}@nil%
  \fi
\fi
\bbl@xin{/e}{/\bbl@cl{lnbrk}}%
\ifin@else\bbl@xin{/k}{/\bbl@cl{lnbrk}}\fi
\bbl@arabicjust
\bbl@xin{/p}{/\bbl@cl{lnbrk}}%
\AtBeginDocument{\@nameuse{bbl@tibetanjust}}%
% == Line breaking: hyphenate.other.(locale|script) ==
\ifx\bbl@lbkflag@empty\else
  \bbl@xin{\bbl@pr@hyotl{\languagename}}%
  \bbl@csarg\replace{\bbl@pr@hyotl{\languagename}}{%}
  \bbl@startcommands*{\languagename}{}
  \bbl@csarg\bbl@foreach{\bbl@pr@hyotl{\languagename}}{%
    \ifcase\bbl@engine
      \ifnum##1<257
        \SetHyphenMap{\BabelLower{##1}{##1}}%
      \else
        \SetHyphenMap{\BabelLower{##1}{##1}}%
      \ifnum##1<257
        \global\lccode##1=##1\relax
      \else
        \global\lccode##1=##1\relax
      \fi}
    \bbl@endcommands%
  \bbl@ifunset{\bbl@pr@hyots{\languagename}}{%}
  \bbl@csarg\replace{\bbl@pr@hyots{\languagename}}{%}
  \bbl@startcommands*{\languagename}{}
  \bbl@csarg\bbl@foreach{\bbl@pr@hyots{\languagename}}{%
    \ifcase\bbl@engine
      \ifnum##1<257
        \global\lccode##1=##1\relax
      \else
        \global\lccode##1=##1\relax
      \fi}
    \bbl@endcommands%
  \bbl@ifunset{\bbl@pr@hyots{\languagename}}{%}
  \bbl@csarg\replace{\bbl@pr@hyots{\languagename}}{%}
  \bbl@startcommands*{\languagename}{}
  \bbl@csarg\bbl@foreach{\bbl@pr@hyots{\languagename}}{%
    \ifcase\bbl@engine
      \ifnum##1<257
        \global\lccode##1=##1\relax
      \else
        \global\lccode##1=##1\relax
      \fi}
    \bbl@endcommands%
  \fi
% == Counters: maparabic ==
% Native digits, if provided in ini (TeX level, xe and lua)
\ifcase\bbl@engine\else
  \fi
59
\ifx\bbl@KVP@casing\@nnil\else
\bbl@csarg\xdef{casing@\languagename}\{\@nameuse{bbl@casing@\languagename}-x-\bbl@KVP@casing}\fi
\% == Calendars ==
\ifx\bbl@KVP@calendar\@nnil\else
\edef\bbl@KVP@calendar\{\bbl@cl{calpr}\}\fi
\% == engine specific extensions ==
\% Defined in XXXbabel.def
\% == require.babel in ini ==
\ifx\bbl@beforestart\relax\else % But not in doc aux or body
\bbl@ifunset{bbl@rqtex@\languagename}\{}%
Depending on whether or not the language exists (based on \date<language>), we define two macros. Remember \bbl@startcommands opens a group.

\def\bbl@provide@new#1{%
  \@namedef{date#1}{}% marks lang exists - required by \bbl@startcommands
  \@namedef{extras#1}{}%
  \@namedef{noextras#1}{}%
  \bbl@startcommands*{#1}{captions}%
  \ifx\bbl@KVP@captions\@nnil % and also if import, implicit
    \def\bbl@tempb##1{% elt for \bbl@captionslist
      \SetString\##1{\bl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}%}
  \expandafter\bbl@tempb\bbl@captionslist\@empty
  \else
    \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
    \bbl@read@ini{\bbl@initoload}2% % Same
  \fi
\}

\def\bbl@startcommands*#1{\@empty}{\@empty}
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 hyphen rules option is handled with an auxiliary macro. This macro is called in three cases: when a language is first declared with `\babelprovide`, with `\babelprovidemyrules` and with `\import`.
\def\bbl@provide@hyphens#1{% 
\@tempcnta\m@ne % a flag 
\ifx\bbl@KVP@hyphenrules\@nnil\else 
\bbl@replace\bbl@KVP@hyphenrules{ }{,}\
\bbl@foreach\bbl@KVP@hyphenrules{ % if not yet found 
\ifnum\@tempcnta=\m@ne % After a possible + 
\bbl@ifsamestring{##1}{+}% 
\bbl@carg\addlanguage{l@##1}{}% 
\bbl@ifunset{l@##1}% After a possible + 
\@tempcnta\@nameuse{l@##1}% 
\fi}\
\ifnum\@tempcnta=\m@ne
\bbl@warning{% Requested 'hyphenrules' for '\languagename' not found:\% \bbl@KVP@hyphenrules.\% 
Using the default value. Reported}% 
\fi 
\fi 
\ifnum\@tempcnta=\m@ne % if no opt or no language in opt found 
\ifx\bbl@KVP@captions@@\@nnil % TODO. Hackish. See above. 
\bbl@ifunset{bbl@hyphr@#1}{}% use value in ini, if exists 
\bbl@exp{\bl@ifblank{\bbl@cs{hyphr@#1}}}{}% 
\bbl@ifunset{l@bbl@cl{hyphr}}{}% if hyphenrules found: 
\@tempcnta\@nameuse{l@bbl@cl{hyphr}{}% 
\fi 
\fi 
\bbl@ifunset{l@#1}% 
\ifnum\@tempcnta=\m@ne 
\bbl@carg\adddialect{l@#1}\language 
\else 
\bbl@carg\adddialect{l@#1}\@tempcnta 
\fi}\
\bbl@if_unset{l@#1}\
\ifnum\@tempcnta=\m@ne 
\global\bbl@carg\chardef{l@#1}\@tempcnta 
\fi 
\fi}}

The reader of babel-...tex files. We reset temporarily some catcodes.
\def\bbl@input@texini#1{% 
\bbl@bsphack 
\bbl@exp{ % 
\catcode`\%=14 \catcode`\{=2 \lowercase{\InputIfFileExists{babel-#1.tex}{}{}}\% 
\catcode`\%=14 \catcode`\{=2 % use value in ini, if exists 
\bbl@exp{\bbl@ifblank{\bbl@cs{hyphr@#1}}}% 
\bbl@if_unset{l@bbl@cl{hyphr}}% 
\bbl@if_unset{l@bbl@cl{hyphr}{}% 
\fi 
\fi 
\bbl@if_unset{l@#1}% 
\ifnum\@tempcnta=\m@ne 
\bbl@carg\adddialect{l@#1}\language 
\else 
\bbl@carg\adddialect{l@#1}\@tempcnta 
\fi}\
\bbl@if_unset{l@#1}\
\ifnum\@tempcnta=\m@ne 
\global\bbl@carg\chardef{l@#1}\@tempcnta 
\fi 
\fi}}

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 \bbl@provide, with 'slashed' keys. There are 3 steps: first read the ini file and store it; then traverse the stored values, and process some groups if required (date, captions, labels, counters); finally, 'export' some values by defining global macros (identification, typography, characters, numbers). The second argument is 0 when called to read the minimal data for fonts; with \bbl@provide it's either 1 or 2.

\def\bbl@loop@ini{%}
  \loop
    \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
      \endlinechar\m@ne
      \read\bbl@readstream to \bbl@line
      \endlinechar`^^M
      \ifx\bbl@line\@empty\else
        \expandafter\bbl@iniline\bbl@line\bbl@iniline
      \fi
    \repeat

\ifx\bbl@readstream@undefined
\csname newread\endcsname\bbl@readstream
\fi

\def\bbl@loop@ini#1#2{%}
  \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}

% == Store ini data in \bbl@inidata ==
\catcode`\|=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
\bbl@info{Importing \ifcase#2font and identification \or basic \fi
  data for \languagename\%
  (#1: \languagename). Perhaps you misspelled it or your\%
  installation is not complete.)%
  \Fix the name or reinstall babel.\%}
%else
%  % == Process stored data ==
%  \bbl@csarg{xdef\lini@\languagename}\{#1\%
%  \bbl@loop@ini"}
% == 'Export' data ==
\global\bbl@ini@exports{#2}%
\global\let\bbl@inidata@\languagename\bbl@inidata
\global\let\bbl@inidata\@empty
\bbl@exp{\\bbl@add@list\\bbl@ini@loaded{\languagename}}%
\bbl@toglobal\bbl@ini@loaded
\fi
\closein\bbl@readstream}
def\bbl@read@ini@aux{%
def\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@
  \bb@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
\bbl@read@ini, but with some changes
\bbl@startcommands*{#1}{captions}%
% Activate captions/... and modify exports
\bbl@csargs\def{\inikv@captions.licr}##1##2{%
  \setlocalecaption{#1}{##1}{##2}}%
\def\bbl@inikv@captions##1##2{%
\bbl@ini@captions@aux{##1}{##2}}%
\def\bbl@stringdef##1##2{\gdef##1{##2}}%
\def\bbl@exportkey##1##2##3{%
  \bbl@ifunset{\bbl@@kv@##2}{}%
  {\expandafter\ifx\csname \bbl@@kv@##2\endcsname\@empty\else
    \bb@exp{\global\let\bbl@@@kv@##2\empty}\fi}}%
% As with \bbl@read@ini, but with some changes
\bbl@read@ini@aux
\bbl@ini@exports@tw@
% Update inidata@lang by pretending the ini is read.
\def\bbl@elt##1##2##3{%
\def\bbl@section{##1}%
\def\bbl@section{##1}%
\bbl@iniline##2=##3\bbl@iniline}%
c\csname bbl@inidata@#1\endcsname
\global\bbl@csargs\let{\bbl@inidata@#1}\bbl@inidata
\StartBabelCommands*{#1}{date}% And from the import stuff
\bbl@savetoday
\bbl@savedate
\bbl@endcommands}
As a somewhat hackish tool to handle calendar sections. TODO. To be improved.
def\bbl@ini@calendar{#1}%
\lowercase{\def\bbl@tempa{=#1=}}%
\bbl@replace\bbl@tempa{=date.gregorian}{}%
\bbl@replace\bbl@tempa{=date.}{}%
in@{.licr=}#1=
\ifin@
  \bb@ifunset{\bbl@inikv@#1}{%
    \csname \bbl@inikv@#1\endcsname{#2}{#3}}%
\bbl@inidata}
\def\bbl@ini@calendar#1{%
  \lowercase{\def\bbl@tempa{=#1=}}%
  \bbl@replace\bbl@tempa{=date.gregorian}{}%
  \bbl@replace\bbl@tempa{=date.}{}%
  \in@{.licr=}#1=}
  \bb@ifunset{\bbl@inikv@#1}{}%
  \csname \bbl@inikv@#1\endcsname{#2}{#3}}%
\bbl@inidata}
\def\bbl@ini@calendar{#1}%
\lowercase{\def\bbl@tempa{=#1=}}%
\bbl@replace\bbl@tempa{=date.gregorian}{}%
\bbl@replace\bbl@tempa{=date.}{}%
in@{.licr=}#1=
\ifin@
  \bb@ifunset{\bbl@inikv@#1}{}%
  \csname \bbl@inikv@#1\endcsname{#2}{#3}}%
\bbl@inidata}
\def\bbl@ini@calendar{#1}%
\lowercase{\def\bbl@tempa{=#1=}}%
\bbl@replace\bbl@tempa{=date.gregorian}{}%
\bbl@replace\bbl@tempa{=date.}{}%
in@{.licr=}#1=
\ifin@
  \bb@ifunset{\bbl@inikv@#1}{}%
  \csname \bbl@inikv@#1\endcsname{#2}{#3}}%
\bbl@inidata}
\def\bbl@ini@calendar{#1}%
\lowercase{\def\bbl@tempa{=#1=}}%
\bbl@replace\bbl@tempa{=date.gregorian}{}%
\bbl@replace\bbl@tempa{=date.}{}%
in@{.licr=}#1=
\ifin@
  \bb@ifunset{\bbl@inikv@#1}{}%
  \csname \bbl@inikv@#1\endcsname{#2}{#3}}%
\bbl@inidata}
\def\bbl@ini@calendar{#1}%
\lowercase{\def\bbl@tempa{=#1=}}%
\bbl@replace\bbl@tempa{=date.gregorian}{}%
\bbl@replace\bbl@tempa{=date.}{}%
in@{.licr=}#1=
\ifin@
  \bb@ifunset{\bbl@inikv@#1}{}%
  \csname \bbl@inikv@#1\endcsname{#2}{#3}}%
\bbl@inidata}
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.

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

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

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

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

Key-value pairs are treated differently depending on the section in the ini file. The following macros are the readers for identification and typography. Note \bbl@ini@exports is called always (via \bbl@inisec), while \bbl@after@ini must be called explicitly after \bbl@read@ini if necessary. Although BCP 47 doesn’t treat ‘-x-’ as an extension, the CLDR and many other sources do (as a private use extension). For consistency with other single-letter subtags or ‘singletons’, here is considered an extension, too.
\bbl@exp{\\bbl@exportkey\{lname\}\{identification.name.opentype\}\\%  
\csname bbl@elnname\languagename\endcsname}}%  
\bbl@exportkey\{tbcp\}\{identification.tag.bcp47\}{}%  
\bbl@exportkey\{lcasing\}\{identification.language.tag.bcp47\}\{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\}{}%  
\ifbbl@bcptoname\xdef{bcp@map\bbl@cl\{tbcp\}}{\languagename}\\fi%  
\directlua{Babel.locale_props\{\the\bbl@cs\{id@@\languagename\}\}.script\%  
= '\bbl@cl\{sbcp\}'\}%  
\fi%  
\% Conditional\%  
\edef\bbl@iniokv\{#1\#2\})){#2}% 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\}{}%  
\bbl@exportkey\{quote\}\{characters.delimiters.quotes\}{}%  
\bbl@exportkey\{dgnat\}\{numbers.digits.native\}{}%  
\def\bbl@iniokv\#1\#2\{\def\toks@\{#2\} \xdef\@kv@{\bbl@section.#1}{\the\toks@}}%  
\let\bbl@iniokv@identification\bbl@iniokv%  
\let\bbl@iniokv@date\bbl@iniokv%  
\let\bbl@iniokv@typography\bbl@iniokv%  
\let\bbl@iniokv@characters\bbl@iniokv%  
\let\bbl@iniokv@numbers\bbl@iniokv%  
\bbl@savestrings%  
% Additive numerals require an additional definition. When \.1 is found,  
% two macros are defined – the basic one, without \.1 called by  
% \localenumeral, and another one preserving the trailing \.1 for the  
% 'units'.
\def\bbl@init\{\@iv\}
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}.

\def\bbl@ini@captions{\begin{template}{\bbl@tempa}{}\end{template}}%
Labels. Captions must contain just strings, no format at all, so there is new group in ini files.

```latex
\def\bbl@list@the{%
\part,chapter,section,subsection,subsubsection,paragraph,\%
subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,\%
table,page,footnote,mpfootnote,mpfn}\def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
\bbl@ifunset{bbl@map@#1@\languagename}{}\else\@nameuse{#1}\fi\bbl@map@#1@\languagename\fi
\def\bbl@inikv@labels#1#2{\in@{.map}{#1}\ifin@%\ifx\bbl@KVP@labels\@nnil\else\bbl@xin@{ map }{ \bbl@KVP@labels\space}\ifin@%\def\bbl@tempc{#1}\bbl@replace\bbl@tempc{.map}{}\bbl@exp{\in@{,#2,}{,arabic,roman,\Roman,\alph,\Alph,fnsymbol,}}\bbl@exp{\gdef\bbl@map@\bbl@tempc @\languagename}{\ifin@\bbl@tempc\else\localecounter{#2}\fi}\bbl@foreach\bbl@list@the{%\bbl@ifunset{the##1}{}\else\bbl@exp{\let\bbl@tempd\the##1}\bbl@exp{\bbl@sreplace\the##1{\bbl@tempc}{\bbl@map@cnt{\bbl@tempc}{##1}}\bbl@sreplace\the##1{\empty\bbl@tempc}{\bbl@map@cnt{\bbl@tempc}{##1}}}\expandafter\ifx\csname the##1\endcsname\bbl@tempd\else\toks@{\the\toks@}\fi%\fi%\fi%\fi\fi%\else%\The following code is still under study. You can test it and make%\% suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's%\% language dependent.%\in@{enumerate.}{#1}\ifin@%\def\bbl@tempa{#1}\bbl@replace\bbl@tempa{enumerate.}{}\def\bbl@toreplace{#2}\bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%\nobreakspace{}}%\fi\fi%\fi%\else%\The following code is still under study. You can test it and make%\% suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's%\% language dependent.%\in@{enumerate.}{#1}\ifin@%\def\bbl@tempa{#1}\bbl@replace\bbl@tempa{enumerate.}{}\def\bbl@toreplace{#2}\bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%\fi\fi%\else%
```
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.

\def\bbl@chaptype{chapter}
\ifx\@makechapterhead\@undefined
\let\bbl@patchchapter\relax
\else\ifx\thechapter\@undefined
\let\bbl@patchchapter\relax
\else\ifx\ps@headings\@undefined
\let\bbl@patchchapter\relax
\else
\def\bbl@patchchapter{\
global\let\bbl@patchchapter\relax\
gdef\bbl@chfmt{\
@chappapp\space\thechapter}\
\bbl@ifunset{bbl@bbl@chaptype fmt@\languagename}{}{bbl@bbl@chaptype fmt@\languagename}}
\bbl@sreplace\ps@headings{\@chapapp\ 
thechapter}{\bbl@chfmt}
\bbl@sreplace\chaptermark{\@chapapp\ 
thechapter}{\bbl@chfmt}
\bbl@sreplace\@makechapterhead{\@chapapp\ \nthechapter}{\bbl@chfmt}
\bbl@toglobal\appendix
\bbl@toglobal\ps@headings
\bbl@toglobal\chaptermark
\bbl@toglobal\@makechapterhead
\let\bbl@patchappendix\bbl@patchchapter
\fi\fi\fi
\fi\fi\fi
\fi\fi\fi
\def\bbl@patchpart{%
\global\let\bbl@patchpart\relax
\gdef\bbl@partfmt{%
@chappapp
\partname\ nobreakspace\ thepart}
\bbl@ifunset{bbl@bbl@partfmt@\languagename}{}{bbl@bbl@partfmt@\languagename}}
\bbl@sreplace\@part{\partname\ nobreakspace\ thepart}{\bbl@partfmt}
\bbl@toglobal\@part
\fi
\fi
\fi
\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@calpr}}
\edef\bbl@thempr{#2}
\edef\bbl@themp{#3}
\edef\bbl@themp2{#3}
\edef\bbl@theympr{#2}
\edef\bbl@theymp{#3}
\edef\bbl@theymp2{#3}
\edef\bbl@the{#4}
\edef\bbl@tempe{bbl@calpr@\languagename}}
\endgroup

\begin{document}
\section{Introduction}

\date{\today}
\begin{abstract}

\end{abstract}

\maketitle

\section{Related Work}

\section{Conclusion}

\end{document}
% Hackish
\let\bbl@calendar\@empty
\let\bbl@variant\@empty
\let\bbl@convert\relax
\def\bbl@tempb##1=##2\@@{% 
\@namedef{bbl@ld@##1}{##2}}%
\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}%
\fi\fi
\@nameuse{bbl@precalendar}% Remove, eg, +, -civil (-ca-islamic)
\edef\bbl@calendar{% Used in \month..., too
\bbl@ld@calendar
\ifx\bbl@ld@variant\@empty\else
.\bbl@ld@variant
\fi}%
\bbl@cased
{\@nameuse{bbl@date@\languagename @\bbl@calendar}%
\bbl@they\bbl@them\bbl@thed}%
\endgroup}

eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
\def\bbl@inidate#1.#2.#3.#4elax#5#6{% TODO - ignore with 'captions'
\bbl@trim\def\bbl@tempa{#1.#2}%
\bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
{\bbl@trim\def\bbl@tempa{#3}%
\bbl@trim\toks@{#5}%
\@temptokena\expandafter{\bbl@savedate}%
\bbl@exp{ Reverse order - in ini last wins
\def\\\bbl@savedate{%
\\\SetString<month><roman numeral>\bbl@tempa#6 name>{\the\toks@%}
\the\@temptokena})}
\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
{\lowercase{\def\bbl@tempb{#6}}%}
\bbl@trim\def\bbl@toreplace{#5}%
\bbl@TG@@date
\global\bbl@csarg\let{date@\languagename @\bbl@tempb}\bbl@toreplace
\ifx\bbl@savetoday\@empty
\bbl@exp{% TODO. Move to a better place.
\{\AfterBabelCommands{%
\def\\\languagename date>{\\protect\\\languagename date >}%
\\newcommand\\languagename date >[4]{}%
\\bbl@usedategrouptrue
\\bbl@ensure\\languagename>{% 
\\\SetString\\today{%
\\\languagename date>[convert]{}
\\\the\year\\\the\month\\\the\day})}}%
\fi%}

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).
\let\bbl@calendar\@empty
\newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{\@nameuse{bbl@ca@#2}#1\@}
\newcommand\BabelDateSpace{\nobreakspace}
\newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
\newcommand\BabelDated[1][{{\number#1}}]
\newcommand\BabelDatedd[1][{{\ifnum#1<10 0\fi\number#1}}]
\newcommand\BabelDateM[1][{{\number#1}}]
\newcommand\BabelDateMM[1][{{\ifnum#1<10 0\fi\number#1}}]
\newcommand\BabelDateMMMM[1][{{\csname month\romannumeral#1\strut\bbl@calendar name\endcsname}}]
\newcommand\BabelDatey[1][{{\number#1}}]
\newcommand\BabelDateyy[1][{{\ifnum#1<10 0\number#1 \else\ifnum#1<100 \number#1 \else\ifnum#1<1000 \expandafter\@gobble\number#1 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 \else\bbl@error{Currently two-digit years are restricted to the range 0-9999.} \else\fi\fi\fi\fi}}]
\newcommand\BabelDateyyyy[1][{{\number#1}}]
\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{####2}}\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@xdatecntr{\expandafter\bbl@xdatecntr\expandafter}
\let\bbl@release@transforms\@empty
\bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
\bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
\def\bbl@transforms@aux#1#2#3#4,#5\relax{#1[#2]{#3}{#4}{#5}}
\begingroup % A hack. TODO. Don't require an specific order
\catcode`%=12
\catcode`&=14
\gdef\babeltempa{}&%
\global\let\babeltempc{}&%
\bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}
\endgroup
Transforms.
\let\bbl@release@transforms\empty
\bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
\bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
\def\bbl@transforms\aux#1#2#3#4,#5\relax{#1[#2]{#3}{#4}{#5}}
\begingroup % A hack. TODO. Don't require an specific order
\catcode`\%=12
\catcode`\&=14
\gdef\babeltempa{}&%
\directlua{
  local str = [=[#2]=] %
  str = str:gsub('%\d+%\d+%d+$', '')
  token.set_macro('babeltempa', str)
}&%
\def\babeltempc{}&%
\bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
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@csarg{\bbl@cs{lname@#1}}\{\bbl@cs{sname@#1}}\{Default}\}\bbl@csarg{\bbl@cs{sname@#1}}\{\bbl@cs{soft@#1}\}{DFLT}\}\bbl@csarg{\bbl@cs{ lname@#1}}\bbl@csarg{\bbl@cs{dirname@#1}}\bbl@csarg{\bbl@cs{language@#1}}\bbl@csarg{\bbl@cs{script@#1}}\bbl@csarg{\bbl@cs{engine@#1}}\bbl@csarg{\bbl@cs{prevh@#1}}\bbl@csarg{\bbl@cs{prevc@#1}}\bbl@csarg{\bbl@cs{prehd}@undefined\global\let\bbl@xenohyph\bbl@xenohyph@d\ifx\AtBeginDocument\@notprerr\expandafter\@secondoftwo % to execute right now\fi\AtBeginDocument{\bbl@patchfont{\bbl@xenohyph}}\bbl@release@transforms{, ,}{\bbl@release@transforms{, ,}{}}\bbl@release@transforms{, ,}{}}
The following ini reader ignores everything but the identification section. It is called when a font is defined (ie, when the language is first selected) to know which script/language must be enabled. This means we must make sure a few characters are not active. The ini is not read directly, but with a proxy tex file named as the language (which means any code in it must be skipped, too).

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.

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

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

The information in the identification section can be useful, so the following macro just exposes it with a user command.
LaTeX needs to know the BCP 47 codes for some features. For that, it expects \texttt{BCPdata} to be defined. While language, region, script, and variant are recognized, extension.⟨⟩ for singletons may change.

\providecommand\BCPdata{}  
\ifx\renewcommand\@undefined\else % For plain. TODO. It’s a quick fix  
\renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty}  
\def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%  
\@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}{\bbl@bcpdata@ii{#6}\languagename}{\bbl@bcpdata@ii{#1#2#3#4#5#6}\languagename}}%  
\def\bbl@bcpdata@ii#1#2{%  \bbl@ifunset{bbl@info@#1.tag.bcp47}{\bbl@error{Unknown field '#1' in \string\BCPdata.\% Perhaps you misspelled it.\% (See the manual for details.)}}{%  \bbl@ifunset{bbl@cs{bbl@info@#1.tag.bcp47\#2}}{}{\bbl@cs{bbl@info@#1.tag.bcp47\#2}}}}\fi%  Still somewhat hackish. WIP.  
\namedef{bbl@localeinfo}{\texttt{localeinfo}}  
\namedef{bbl@error}{\texttt{error}}  
\namedef{bbl@nothing}{\texttt{nothing}}  
\namedef{bbl@load@info}{\texttt{load@info}}  
\def\bbl@load@info#1{%  \bbl@ifundefined{bbl@info@#1}{\bbl@error{I’ve found no info for the current locale.\% The corresponding ini file has not been loaded\% Perhaps it doesn’t exist\% (See the manual for details.)}}{%}
5 Adjusting the Babel behavior

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

\newcommand\babeladjust[1]{% TODO. Error handling.
\@ifstar\@bbladjust@star{#1}{% 
\ifvmode
\ifnum\currentgrouplevel=\z@
\directlua{ Babel.#1 }%
\fi
\fi
\@gobble
\ifx\relax
{\string#1 will be set to \relax}%
{Perhaps you misspelled it.}%
}\fi
\let\@bbladjust\@empty
\newcommand\LocaleForEach{\@foreach\bbl@ini@loaded}

\newcommand\getlocaleproperty{\bbl@getproperty@s\bbl@getproperty@x}
\def\bbl@getproperty@s#1#2#3{\let#1\relax
\def\bbl@elt##1##2##3{\if\string##1/\string##2\string#3\fi
\def\bbl@elt####1####2####3{}}%
\bbl@cs{inidata@#2}}%
\def\bbl@getproperty@x#1#2#3{\bbl@getproperty@s{#1}{#2}{#3}%
\ifx#1\relax
\@bblerror
{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{\@foreach\bbl@ini@loaded}

\newcommand\getlocaleproperty{%
\@ifstar\@bblgetproperty@star{#1}{% 
\ifvmode
\ifnum\currentgrouplevel=\z@
\directlua{ Babel.#1 }%
\fi
\fi
\@gobble
\ifx\relax
{\string#1 will be set to \relax}%
{Perhaps you misspelled it.}%
\fi
\let\@bblgetproperty\@empty
\newcommand\LocaleForEach{\@foreach\bbl@ini@loaded}

3621 \@namedef{bbl@ADJ@bidi.math@on}{% 3622 \let\bbl@noamsmath@empty} 3623 \@namedef{bbl@ADJ@bidi.math@off}{% 3624 \let\bbl@noamsmath@relax} 3625 \@namedef{bbl@ADJ@bidi.mapdigits@on}{% 3626 \bbl@adjust@lua{bidi}{digits_mapped=true}} 3627 \@namedef{bbl@ADJ@bidi.mapdigits@off}{% 3628 \bbl@adjust@lua{bidi}{digits_mapped=false}} 3629 \% 3630 \@namedef{bbl@ADJ@linebreak.sea@on}{% 3631 \bbl@adjust@lua{linebreak}{sea_enabled=true}} 3632 \@namedef{bbl@ADJ@linebreak.sea@off}{% 3633 \bbl@adjust@lua{linebreak}{sea_enabled=false}} 3634 \@namedef{bbl@ADJ@linebreak.cjk@on}{% 3635 \bbl@adjust@lua{linebreak}{cjk_enabled=true}} 3636 \@namedef{bbl@ADJ@linebreak.cjk@off}{% 3637 \bbl@adjust@lua{linebreak}{cjk_enabled=false}} 3638 \@namedef{bbl@ADJ@justify.arabic@on}{% 3639 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}} 3640 \@namedef{bbl@ADJ@justify.arabic@off}{% 3641 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}} 3642 \% 3643 \def\bbl@adjust@layout#1{% 3644 \ifvmode 3645 #1% 3646 \expandafter\@gobble 3647 \fi 3648 {\bbl@error % The error is gobbled if everything went ok. 3649 {Currently, layout related features can be adjusted only\% 3650 in vertical mode.}% 3651 {Maybe things change in the future, but this is what it is.}}}% 3652 \@namedef{bbl@ADJ@layout.tabular@on}{% 3653 \ifnum\bbl@tabular@mode=\tw@ 3654 \bbl@adjust@layout{\let\tabular\bbl@NL@@tabular}% 3655 \else 3656 \chardef\bbl@tabular@mode@one 3657 \fi}% 3658 \@namedef{bbl@ADJ@layout.tabular@off}{% 3659 \ifnum\bbl@tabular@mode=\tw@ 3660 \bbl@adjust@layout{\let\tabular\bbl@OL@@tabular}% 3661 \else 3662 \chardef\bbl@tabular@mode@two 3663 \fi}% 3664 \@namedef{bbl@ADJ@layout.lists@on}{% 3665 \bbl@adjust@layout{\let\list\bbl@NL@list}} 3666 \@namedef{bbl@ADJ@layout.lists@off}{% 3667 \bbl@adjust@layout{\let\list\bbl@OL@list}} 3668 \% 3669 \@namedef{bbl@ADJ@autoload.bcp47@on}{% 3670 \bbl@bcpallowedtrue} 3671 \@namedef{bbl@ADJ@autoload.bcp47@off}{% 3672 \bbl@bcpallowedfalse} 3673 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{% 3674 \def\bbl@bcp@prefix{#1}} 3675 \@namedef{bbl@ADJ@autoload.options}#1{% 3676 \def\bbl@autoload@options{#1}} 3677 \let\bbl@autoload@bcpoptions\@empty 3678 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{% 3679 \def\bbl@autoload@bcpoptions{#1}} 3680 \newif\iffbl@bcptoname 3681 \@namedef{bbl@ADJ@bcp47.toname@on}{% 3682 \bbl@bcptonametrue 3683 \bbl@bcptonametrue
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.

\begin{verbatim}
\DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
\DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
\DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
\DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
\end{verbatim}

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

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

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

3781 \bbl@xin@{B}\bbl@opt@safe
3782 \ifin@
3783 \bbl@redefine\citex[#1]#2{%
3784 \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
3785 \org@citex[#1]({\@tempa})%
3786 }{}

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

3786 \AtBeginDocument{%
3787 \@ifpackageloaded{natbib}{% 

3788 \def\citex[#1][#2][#3]{% 
3789 \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse 
3790 \org@citex[#1][#2]{{\@tempa}}% 
3791 }{}

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

3792 \AtBeginDocument{%
3793 \@ifpackageloaded{cite}{% 
3794 \def\citex[#1]#2{% 
3795 \@safe@activetrue\org@citex[#1][#2]\@safe@activesfalse% 
3796 }{}%

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

3797 \bbl@redefine\nocite#1{% 
3798 \@safe@activetrue\org@nocite#1\@safe@activesfalse

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

3799 \bbl@redefine\bibcite{%
3800 \bbl@cite@choice
3801 \bibcite}

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

3802 \def\bibcite#1#2{% 
3803 \org@bibcite#1{\@safe@activesfalse#2}}

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

3804 \def\bibcite#1{% 
3805 \global\let\bibcite@bibcite\bbl@bibcite
3806 \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}% 
3807 \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}% 
3808 \global\let\bbl@cite@choice\relax}
When a document is run for the first time, no .aux file is available, and \bibcite will not yet be properly defined. In this case, this has to happen before the document starts.

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

3809 \AtBeginDocument{\bbl@cite@choice}
3810 \@bibitem
3811 \@safe@activestrue\org@\bibitem[#1]\@safe@activestfalse
3812 \else
3813 \let\org@nocite\nocite
3814 \let\org@citex@citetex
3815 \let\org@bibcite\bibcite
3816 \let\org@\bibitem\@bibitem
3817 \fi

5.2 Marks
\markright Because the output routine is asynchronous, we must pass the current language attribute to the head lines. To achieve this we need to adapt the definition of \markright and \markboth somewhat. However, headlines and footlines can contain text outside marks; for that we must take some actions in the output routine if the 'headfoot' options is used. We need to make some redefinitions to the output routine to avoid an endless loop and to correctly handle the page number in bidi documents.

3818 \bbl@trace{Marks}
3819 \IfBabelLayout{sectioning}{
3820 \iffx\bbl@opt@headfoot@nil
3821 \g@addto@macro\@resetactivechars{\
3822 \set@typeset@protect\expandafter\select@language@x\expandafter{\bbl@main@language}\%
3823 \let\protect\noexpand\@expandafter\selectlanguage@x\expandafter{\bbl@main@language}\%
3824 \let\protect\noexpand\@noexpand\selectlanguage@x\expandafter{\bbl@main@language}\%
3825 \ifcase\bbl@bidimode\else \selectlanguage@x{\selectlanguage{\languagename}}%\selectlanguage{\languagename}%
3826 \edef\thepage{\noexpand\babelsublr{\unexpanded\expandafter\noexpand}}%
3827 \fi\%
3828 \fi}
3829 \else
3830 \iff\bbl@opt@headfoot\else
3831 \bbl@ifunset{markright }\bbl@redefine\bbl@redefinerobust
3832 \markright#1\%
3833 \bbl@ifblank{#1}{}\%
3834 \{\org@markright{}\%
3835 \{\toks@{#1}\%
3836 \bbl@exp\%
3837 \{\org@markright{}\protect\foreignlanguage{\languagename}\%
3838 \{\protect\bbl@restore@actives\the\toks@{}\}}
3839 \fi\%
3840 \fi}
3841 \else
3842 \bbl@ifunset{markboth }\bbl@redefine\bbl@redefinerobust
3843 \markboth#1\#
3844 \bbl@ifblank{#1}{}\#
3845 \{\org@markboth{}\protect\foreignlanguage{\languagename}\%
3846 \{\protect\bbl@restore@actives\the\toks@{}\}%
3847 \bbl@ifblank{#1}{}\%
3848 \{\toks@{}\}\

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

3839 \iffx\@mkboth\markboth
3840 \def\bbl@tempc{\let\@mkboth\markboth}%
3841 \else
3842 \def\bbl@tempc{\}
3843 \fi
3844 \bbl@ifunset{markboth }\bbl@redefine\bbl@redefinerobust
3845 \markboth#1\#
3846 \protect@edef\bbl@tempc#1\%
3847 \protect\foreignlanguage{\languagename}\%
3848 \{\protect\bbl@restore@actives\the\toks@{}\}%
3849 \bbl@ifblank{#1}{}\%
3850 \{\toks@{}\}%
5.3 Preventing clashes with other packages

5.3.1 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:

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

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

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

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

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

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

\AtEndOfPackage{\AtBeginDocument{%\ifpackageloaded{hhline}\else\makeatletter\def\@currname{hhline}\input{hhline.sty}\makeatother\fi%\}}

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

\verb|\lowercase{\immediate\openout15=#1#2.fd|\relax\%\immediate\write15{\string\ProvidesFile{#1#2.fd}[\the\year/\two@digits{\the\month}/\two@digits{\the\day} generated font description file]^^J\string\DeclareFontFamily{#1}{#2}{}^^J\string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}{}^^J\string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}{}^^J\string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}{}^^J\string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}{}^^J\string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}{}^^J\string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}{}^^J\string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}{}^^J\string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}{}^}\string\CloseOut15\}\onlypreamble\substitutefontfamily

## 5.4 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of \TeX and \LaTeX always come out in the right encoding. There is a list of non-ASCII encodings. Requested encodings are currently stored in \@fontenc@load@list. If a non-ASCII has been loaded, we define versions of

\index{encoding!non-ASCII}
\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

\newcommand\BabelNonASCII{LGR, LGI, X2, OT2, OT3, OT6, LHE, LWN, LMA, LMC, LMS, LMU}
\newcommand\BabelNonText{TS1, T3, TS3}
\let\org@TeX\TeX
\let\org@LaTeX\LaTeX
\let\ensureascii\@firstofone
\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{% LGR loaded in a non-standard way
\bbl@xin@{,#1,}{,\BabelNonASCII,}
\ifin@
\def\bbl@tempb{#1}% Store last non-ascii
\else\bbl@xin@{,#1,}{,\BabelNonText,}% Pass
\ifin@\else\bbl@xin@{,#1,}{,\BabelNonASCII,}% Store last non-ascii
\fi}%
\ifx\bbl@tempb\@empty\else
\bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
\ifin@\else\edef\bbl@tempc{\cf@encoding}% The default if ascii wins
\fi
\fi
\let\bbl@tempc\bbl@tempb
\renewcommand\ensureascii[1]{\fontencoding{\asciiencoding}\selectfont#1}%%
\DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%%
\DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
\fi}

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

\latinencoding

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

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

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

\AtBeginDocument{%
\@ifpackageloaded{fontspec}%
{\xdef\latinencoding\%
{\ifx\UTFencname\undefined
EU\ifcase\bbl@engine\or2\or1\fi
\else
\UTFencname
\fi}%
{\gdef\latinencoding{OT1}%
{\ifx\cf@encoding\bbl@t@one
{\xdef\latinencoding{\bbl@t@one}%
85
}
\latintext Then we can define the command \latintext which is a declarative switch to a latin font-encoding. Usage of this macro is deprecated.

\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.

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

5.5 Basic bidi support

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

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

\textlatin 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 rl babel did), and by introducing a “middle layer” just below the user interface (sectioning, footnotes).

\textlatin • pdftex provides a minimal support for bidi text, and it must be done by hand. Vertical typesetting is not possible.

\textlatin • xetex is somewhat better, thanks to its font engine (even if not always reliable) and a few additional tools. However, very little is done at the paragraph level. Another challenging problem is text direction does not honour \TeX\ grouping.

\textlatin • luatex can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but bidi text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As Lua\TeX\-ja shows, vertical typesetting is possible, too.
Now come the macros used to set the direction when a language is switched. First the (mostly) common macros.

\bbl@trace{Macros to switch the text direction}
\def\bbl@alscripts{Arabic,Syriac,Thaana,}
\def\bbl@rscripts{Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,\ldots}
\def\bbl@provide@dirs#1{\bbl@xin{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}\
\ifin@\global\bbl@csarg\chardef{wdir@#1}@ne\else\global\bbl@csarg\chardef{wdir@#1}@w\fi\ifodd\bbl@engine\else\bbl@xebidipar\fi}

\bbl@trace{The (mostly) common macros
\@csarg\ifcase{wdir}@#1\%
\directlua{ Babel.locale_props[@the\localeid].textdir = 'l' }\%
\or
\directlua{ Babel.locale_props[@the\localeid].textdir = 'r' }\%
\or
\directlua{ Babel.locale_props[@the\localeid].textdir = 'al' }\%
\fi
\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 %
\bbl@bodydir{#1}%
\bbl@pardir{#1}% <- Must precede \bbl@textdir
\fi
\bbl@textdir{#1}}
% TODO. Only if \bbl@bidimode > 0?:
\AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
\DisableBabelHook{babel-bidi}
Now the engine-dependent macros. TODO. Must be moved to the engine files.
\ifodd\bbl@engine % luatex=1
\else % pdftex=0, xetex=2
\newcount\bbl@dirlevel
\chardef\bbl@thetextdir=z@
\chardef\bbl@thepardir=z@
\def\bbl@textdir#1{%
\ifcase#1\relax
\chardef\bbl@thetextdir=z@
\@nameuse{setlatin}%
\bbl@textdir@i\beginL\endL
\else
\chardef\bbl@thetextdir=1
\@nameuse{setnonlatin}%
\bbl@textdir@i\beginR\endR
\fi}
\def\bbl@textdir@i#1#2{%
\ifhmode
\ifnum\currentgrouplevel>\z@
\ifnum\currentgrouplevel=\bbl@dirlevel
\bbl@error{Multiple bidi settings inside a group}%
{I'll insert a new group, but expect wrong results.}%
\begingroup\aftergroup\aftergroup\aftergroup\aftergroup\egroup
\else
\ifcase\currentgrouptype\or % 0 bottom
\aftergroup\aftergroup\aftergroup\aftergroup\egroup
% 1 simple {} \\
\or
\aftergroup\aftergroup\aftergroup\aftergroup\egroup
% 2 hbox \\
\or
\aftergroup\aftergroup\aftergroup\aftergroup\egroup
% 3 adj hbox \\
\or
\aftergroup\aftergroup\aftergroup\aftergroup\egroup
% 7 noalign \\
\or
\aftergroup\aftergroup\aftergroup\aftergroup\egroup
% output math disc insert vcent mathchoice \\
\or
\aftergroup\aftergroup\aftergroup\aftergroup\egroup
% 14 \begingroup \\
\else
\aftergroup\aftergroup\aftergroup\aftergroup\egroup
% 15 adj \\
\fi
\fi
\bbl@dirlevel\currentgrouplevel
The following command is executed only if there is a right-to-left script (once). It activates the \everypar hack for \texttt{xetex}, to properly handle the par direction. Note text and par dirs are decoupled to some extent (although not completely).

\def\bbl@xeeverypar{\ifcase\bbl@thepardir\ifcase\bbl@thetextdir\else\beginR\fi\else\fi\fi}\let\bbl@severypar\everypar\newtoks\everypar\everypar=\bbl@severypar\bbl@severypar{\bbl@xeeverypar\the\everypar}\ifnum\bbl@bidimode>200 % Any xe bidi=
\let\bbl@textdir@i\@gobbletwo\let\bbl@xebidipar\@empty\AddBabelHook{bidi}{foreign}{\def\bbl@tempa{\def\BabelText####1}\ifcase\bbl@thetextdir\expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}\else\expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}\fi}\def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}\fi

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

\DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}\AtBeginDocument{\ifx\pdfstringdefDisableCommands\@undefined\else\ifx\pdfstringdefDisableCommands\relax\else\pdfstringdefDisableCommands{\let\babelsublr\@firstofone}\fi\fi}

\section*{5.6 Local Language Configuration}
\texttt{\loadlocalcfg} At some sites it may be necessary to add site-specific actions to a language definition file. This can be done by creating a file with the same name as the language definition file, but with the extension \texttt{.cfg}. For instance the file \texttt{norsk.cfg} will be loaded when the language definition file \texttt{norsk.ldf} is loaded. For plain-based formats we don't want to override the definition of \texttt{\loadlocalcfg} from \texttt{plain.def}.
5.7 Language options

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

\bbl@trace{Language options}
\let\bbl@afterlang\relax
\let\BabelModifiers\relax
\let\bbl@loaded\@empty
\def\bbl@load@language#1{%
  \InputIfFileExists{#1.ldf}%
  {\edef\bbl@loaded{\CurrentOption}
   \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}
  \expandafter\let\expandafter\bbl@afterlang
    \csname\CurrentOption.ldf-h@@k\endcsname
  \expandafter\let\expandafter\BabelModifiers
    \csname bbl@mod@\CurrentOption\endcsname
  \bbl@exp{\\AtBeginDocument{%
    \bbl@usehooks@lang{\CurrentOption}{begindocument}{{\CurrentOption}}}}%
  \bbl@error{%
    Unknown option '{\CurrentOption}'. Either you misspelled it\%
    or the language definition file '{\CurrentOption}.ldf' was not found}{%}
    {Valid options are, among others: shorthands=, KeepShorthandsActive,\%
    activeacute, activegrave, noconfigs, safe=, main=, math=\%
    headfoot=, strings=, config=, hyphenmap=, or a language name.}}%
}\bbl@try@load@lang#1#2#3{%
  \IfFileExists{\CurrentOption.ldf}%
    {\bbl@load@language{\CurrentOption}}%
  {#1\bbl@load@language{#2}#3}}%
\DeclareOption{hebrew}{{
  \input{rlbabel.def}%
  \bbl@load@language{hebrew}}%
\DeclareOption{hungarian}{{
  \bbl@try@load@lang{}{magyar}{}}}%
\DeclareOption{lowersorbian}{{
  \bbl@try@load@lang{}{lsorbian}{}}}%
\DeclareOption{northernsami}{{
  \bbl@try@load@lang{}{samin}{}}}%
\DeclareOption{nynorsk}{{
  \bbl@try@load@lang{}{norsk}{}}}%
\DeclareOption{polutonikogreek}{{
  \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}%
\DeclareOption{russian}{{
  \bbl@try@load@lang{}{russianb}{}}}%
\DeclareOption{scottishgaelic}{{
  \bbl@try@load@lang{}{scottish}{}}}%
\DeclareOption{ukrainian}{{
  \bbl@try@load@lang{}{ukraineb}{}}}%
\DeclareOption{uppersorbian}{{
  \bbl@try@load@lang{}{usorbian}{}}}%
\begin{itemize}
  \item 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.
\end{itemize}
\ifx\bbl@opt@config\@nnil
  \@ifpackagewith{babel}{noconfigs}{}%
  {\InputIfFileExists{bblopts.cfg}%
    {\typeout{*************************************************************************
      \Local config file bblopts.cfg used^^J%
      *)%}
    {}}}%
\else
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 \texttt{bb@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 \texttt{ldf} and there is no main key. In the latter case (\texttt{bb@opt@main} is still \texttt{@nil}), 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 \texttt{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 correspondin file exists.

\begin{verbatim}
    \InputIfFileExists{bb@opt@config.cfg}\
    \PackageWarning{\texttt{bb@opt@config.cfg} used^^J\texttt{\bb@opt@config.cfg}}\
    \PackageError{Local config file \texttt{\bb@opt@config.cfg} not found}{Perhaps you misspelled it.}\
    \fi

    \InputIfFileExists{babel-#1.tex}{\def{bb@opt@main}{#1}}\
    \IfFileExists{#1.ldf}{\def{bb@opt@main}{#1}}\
    \fi\
    \fi% \texttt{main} is still \texttt{@nil}
\end{verbatim}
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):

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

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

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

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 \texttt{\textbackslash babelprovides} can’t go inside a \texttt{\textbackslash DeclareOption}; this explains why it’s executed directly, with a dummy declaration. Then all languages have been loaded, so we deactivate \texttt{\textbackslash AfterBabelLanguage}.

```latex
\bbl@trace{Option 'main'}
\ifdef\bbl@opt@main\@nnil
\edef\bbl@tempa{\@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}\ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
\def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
\expandafter\bbl@tempa\bbl@loaded,\@nnil
\ifodd\bbl@iniflag % case 1,3 (main is ini)
\bbl@ldfinit
\let\CurrentOption\bbl@opt@main
\bbl@exp{% \bbl@opt@provide = empty if *
\\\bbl@provides[\bbl@opt@provide,import,main]{\bbl@opt@main}{%\bbl@afterldf{}}%
\DeclareOption{\bbl@opt@main}{}%}
\else % case 0,2 (main is ldf)
\iftfl\bbl@loadmain\relax
\DeclareOption{\bbl@opt@main}{\bbl@loadlanguage{\bbl@opt@main}}
\else
\DeclareOption{\bbl@opt@main}{\bbl@loadmain}
```
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.

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.

### 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

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

### 7 Loading hyphenation patterns

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

```\ProvidesFile{hyphen.cfg}[[\jobname]\ \V{\version}] Babel hyphens]
exdef\bbl@format{\jobname}
exdef\bbl@version{\version}
exdef\bbl@date{\jobname}
\ifx\AtBeginDocument\@undefined
\def\empty{}
\fi
```

```\Make sure ProvidesFile is defined\```

```\ProvidesFile{hyphen.cfg}[[\jobname]\ V{\version}] Babel hyphens]
exdef\bbl@format{\jobname}
exdef\bbl@version{\version}
exdef\bbl@date{\jobname}
\ifx\AtBeginDocument\@undefined
\def\empty{}
\fi
```

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

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

This macro takes care of the lines which start with an `=`. It needs an empty token register to begin with. `%bbl@languages` is also set to empty.

```
\toks@{}
\def\bbl@languages{}
```

When no languages have been loaded yet, the name following the `=` will be a synonym for hyphenation register 0. So, it is stored in a token register and executed when the first pattern file has been processed. (The `\relax` just helps to the `\if` below catching synonyms without a language.) Otherwise the name will be a synonym for the language loaded last.

We also need to copy the hyphenmin parameters for the synonym.

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

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

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

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

Pattern files may contain assignments to `\lefthyphenmin` and `\righthyphenmin`. \TeX{} does not keep track of these assignments. Therefore we try to detect such assignments and store them in the `\langle lang⟩hyphenmins` macro. When no assignments were made we provide a default setting. Some pattern files contain changes to the `\lccode` en `\uccode` arrays. Such changes should remain local to the language; therefore we process the pattern file in a group; the `\patterns` command acts globally so its effect will be remembered.

Then we globally store the settings of `\lefthyphenmin` and `\righthyphenmin` and close the group. When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

`\bbl@languages` saves a snapshot of the loaded languages in the form `\bbl@elt{⟨language-name⟩}{{(number)} { (patterns-file)}{(exceptions-file)}}`. Note the last 2 arguments are empty in ‘dialects’ defined in `language.dat` with `=`. Note also the language name can have encoding info.

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

```
\def\process@language#1#2#3{%
  \expandafter\addlanguage\csname l@#1\endcsname
```
They macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. It uses delimited arguments to achieve this.

Now, hooks are defined. For efficiency reasons, they are dealt with in a special way. Besides `luatex`, 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.

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

\openin1 = language.dat

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

\def\languagename{english}%
\ifeof1
\message{I couldn't find the file language.dat, I will try the file hyphen.tex}
\input hyphen.tex\relax
\chardef\l@english\z@
\else
\input babel-def
\fi
\else
\input luababel.def
\fi
\openin1 = babel-bbl@format.cfg
\ifeof1
\else
\input babel-bbl@format.cfg\relax
\fi
\closein1
\endgroup
\bbl@hook@loadkernel{switch.def}

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.

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

\loop
  \endlinechar\me
  \read1 to \bbl@line
  \endlinechar``

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.

\if T\ifeof1F\fi T\relax
  \ifx\bbl@line\@empty\else
    \edef\bbl@line{\bbl@line\space\space\space}%
    \expandafter\process@line\bbl@line\relax
  \fi
\repeat

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

\begingroup
  \def\bbl@elt#1#2#3#4{%
    \global\language=#2\relax
    \gdef\languagename{#1}%
    \def\bbl@elt##1##2##3##4{}}%
  \bbl@languages
  \endgroup

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

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

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

\let\bbl@line\undefined
\let\process@line\undefined
\let\process@synonym\undefined
\let\process@language\undefined
\let\bbl@get@enc\undefined
\let\bbl@hyph@enc\undefined
\let\bbl@tempa\undefined
\let\bbl@hook@loadkernel\undefined
\let\bbl@hook@everylanguage\undefined
\let\bbl@hook@loadpatterns\undefined
\let\bbl@hook@loadexceptions\undefined
\let\bbl@hook@loadkernel\undefined
\let\bbl@hook@everylanguage\undefined
\let\bbl@hook@loadpatterns\undefined
\let\bbl@hook@loadexceptions\undefined

Here the code for \LaTeX ends.

8 Font handling with fontspec

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

\DeclareOption{bidi=default}{\chardef\bbl@bidimode=\me}
\DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
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.

\ifx\fontspec\@undefined
\usepackage{fontspec}
\fi

\newcommand\babelfont[2][]{% 1=langs/scripts 2=fam
\bbl@foreach{#1}{%  
\expandafter\ifx\csname date##1\endcsname\relax
\IfFileExists{babel-##1.tex}{\babelprovide{##1}}{}%  
}\edef\bbl@tempa{#1}%  
\def\bbl@tempb{#2}% Used by \bbl@bblfont
\bbl@loadfontspec
\EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
\bbl@bblfont}

\def\babelfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
\bbl@ifunset{\bbl@tempb family}{\bbl@providefam{\bbl@tempb}}{}%  
\bbl@ifblank{\bbl@tempa}{\bbl@csarg\edef\bbl@tempb dflt@{<>{#1}{#2}}% save \bbl@rmdflt@
\bbl@exp{%  
\let<\bbl@bblfont dflt@=\bbl@tempb dflt@%  
\bbl@font@set<\bbl@bblfont family>@%  
\bbl@csarg\edef\bbl@tempb dflt@##1{<>{#1}{#2}}}}}%
\bbl@csarg\edef\bbl@tempb dflt@script{<>{#1}{#2}}}

If the family in the previous command does not exist, it must be defined. Here is how:
\def\babelfont[2][]{% 1=default font, just in case:  
\bbl@ifunset{\bbl@tempb family}{\bbl@providefam{\bbl@tempb family}}}%
\expandafter\bbl@csarg\edef\bbl@tempb dflt@{<>{#1}{#2}}% save \bbl@rmdflt@
\bbl@exp{%  
\let<\bbl@bblfont dflt@=\bbl@tempb dflt@%  
\bbl@font@set<\bbl@bblfont family>@%  
\bbl@csarg\edef\bbl@tempb dflt@##1{<>{#1}{#2}}}}%
The following macro is activated when the hook babel\-fontspec is enabled. But before, we define a macro for a warning, which sets a flag to avoid duplicate them.

\def\bbl@nostdfont#1{\
  \bbl@ifunset{bbl@WFF@\f@family}\
  {\bbl@csarg\gdef{WFF@\f@family}{}% Flag, to avoid dupl warns\
    \bbl@infowarn{The current font is not a babel standard family:\%\
      #1%\
    fontname\%\
    There is nothing intrinsically wrong with this warning, and\%\
    you can ignore it altogether if you do not need these\%\
    families. But if they are used in the document, you should be\%\
    aware 'babel' will not set Script and Language for them, so\%\
    you may consider defining a new family with \string\belfont.\%\
    See the manual for further details about \string\belfont.\%\
    Reported}}\
  }\
\gdef\bbl@switchfont{\
  \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}\
  \bbl@exp{% eg Arabic -> arabic\
    \lowercase{\edef\bbl@tempa{\bbl@cl{sname}}}}\
  \bbl@foreach\bbl@font@fams{\
    \bbl@ifunset{bbl@##1dflt@\languagename}{% (1) language?\
      \bbl@ifunset{bbl@##1dflt@*\bbl@tempa}{% (2) from script?\
        \bbl@ifunset{bbl@##1dflt@}% 2=F - (3) from generic?\
          {}% 123=F - nothing!\
        \bbl@exp{% 3=T - from generic\
          \global\let\bbl@##1dflt@\languagename\bbl@##1dflt@}}%\
      \bbl@exp{% 2=T - from script\
        \global\let\bbl@##1dflt@\languagename\bbl@##1dflt@}}%\
    }% 1=T - language, already defined\
  }\
  }\
\def\bbl@tempa{\bbl@nostdfont{}}% TODO. Don't use \bbl@tempa\
\bbl@foreach\bbl@font@fams{% don't gather with prev for\
    \bbl@ifunset{bbl@##1dflt@\languagename}{% (1) language?\
      \bbl@ifunset{bbl@##1dflt@\bbl@tempa}{% (2) from script?\
        \bbl@ifnull{\bbl@cl{##1dflt@}}{\bbl@add\originalTeX{\
          \bbl@font@rst{\bbl@cl{##1dflt@}}\
          \bbl@font@set\bbl@##1dflt@\languagename}}%\
      }{\bbl@exp{%\bbl@add\bbl@tempa{* \bbl@##1dflt@=@\f@family\\%\
        100\
      }}}}\
  }{}}%\bbl@ifrestoring{}}}{}

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

\ifx\f@family@undefined\else % if latex\
  \ifcase\bbl@engine % if pdftex\
    \let\bbl@cckeckstfonts\relax\
  \else\
    \\def\bbl@cckeckstfonts{%\
      \begin{group}\
        \global\let\bbl@cckeckstfonts\relax\
        \let\bbl@tempa\empty\
        \bbl@foreach\bbl@font@fams{%\
          \bbl@ifnull{\bbl@cl{##1dflt@}}{\% Flag\
            \bbl@exp{\\bbl@add\\bbl@tempa{* \bbl@##1dflt@=@\f@family}}}\
    }\end{group}\
  \fi\
\fi
Now the macros defining the font with fontspec.
When there are repeated keys in fontspec, the last value wins. So, we just place the ini settings at
the beginning, and user settings will take precedence. We must deactivate temporarily \bbl@mapselect
because \selectfont is called internally when a font is defined.
For historical reasons, LATEX can select two different series (bx and b), for what is conceptually a
single one. This can lead to problems when a single family requires several fonts, depending on the
language, mainly because ‘substitutions’ with some combinations are not done consistently –
sometimes bx/sc is the correct font, but sometimes points to b/n, even if b/sc exists. So, some
substitutions are redefined (in a somewhat hackish way, by inspecting if the variant declaration
contains \ssub*).
\def\bbl@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\
% TODO - next should be global?, but even local does its job. I'm
% still not sure -- must investigate:
\def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
\let\bbl@tempe\bbl@mapselect\edef\bbl@tempb{\bbl@stripslash#4/}% Catcodes hack (better pass it).
\bbl@exp{\let\bbl@tempselect\relax\let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below\let\#4\@empty % Make sure \renewfontfamily is valid\let\bbl@exp[\let\\bbl@tempofamily\<\bbl@stripslash#4\space% eg, '\rmfamily'\<keys_if_exist:nF{fontspec-opentype}{Script/\bbl@cl{sname}}%\{\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}\}\<keys_if_exist:nF{fontspec-opentype}{Language/\bbl@cl{lname}}%\{\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}\}\let\\bbl@tempfss@nx\__fontspec_warning:nx\let\\bbl@tempfss@nxx\__fontspec_warning:nxx\let\\bbl@tempfss@nx\__fontspec_warning:nx\\let\\bbl@tempfss@nxx\__fontspec_warning:nxx\let\\bbl@tempfss@nx\__fontspec_warning:nx\\let\\bbl@tempfss@nxx\__fontspec_warning:nxx\let\\bbl@tempfss@nx\__fontspec_warning:nx\\let\\bbl@tempfss@nxx\__fontspec_warning:nxx\let\\bbl@tempfss@nx\__fontspec_warning:nx\\let\\bbl@tempfss@nxx\__fontspec_warning:nxx\let\\bbl@tempfss@nx\__fontspec_warning:nx\\let\\bbl@tempfss@nxx\__fontspec_warning:nxx\let\\bbl@tempfss@nx\__fontspec_warning:nx\\let\\bbl@tempfss@nxx\__fontspec_warning:nxx
\let\exp{\let<\fontspec_warning:nx>\bbl@tempfs@nx\let<\fontspec_warning:nxx>\bbl@tempfs@nxx}\begingroup
#4\xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}\endgroup % TODO. Find better tests:
\let\xin{\string>\string s\string u\string b\string*}{\expandafter\meaning\csname TU/#1/bx/sc\endcsname}\ifin@\global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}\fi\let\xin{\string>\string s\string u\string b\string*}{\expandafter\meaning\csname TU/#1/bx/scit\endcsname}\ifin@\global\bbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}\fi\let#4\bbl@temp@fam\bbl@exp{\let<\bbl@stripslash#4\space>\bbl@temp@pfam}\let\mapselect\bbl@temp@e

font@rst and famrst are only used when there is no global settings, to save and restore de previous families. Not really necessary, but done for optimization.
\def\bbl@font@rst#1#2#3#4{%\def\bbl@font@fams{rm,sf,tt}

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@footnotechanges{%\def\bbl@footnotes{%\@ifnextchar[\bbl@footnotechanges{%\bbl@footnotechanges%

⟨⟨Font selection⟩⟩

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

Now, the code.

\def\BabelStringsDefault{unicode}
\let\xebl@stop\relax
\AddBabelHook{xetex}{encodedcommands}{% 
def\bbl@tempa{#1}% \ifx\bbl@tempa\@empty\XeTeXinputencoding"bytes"\else\XeTeXinputencoding#1\fi\def\xebl@stop{\XeTeXinputencoding"utf8"} % We may override the ini \expandafter\ifx\csname bbl@intsp@languagename\endcsname\@empty\else\expandafter\ifx\bbl@KVP@intraspace\@nnil\expandafter\bbl@intraspace\bbl@KVP@intraspace\@@\fi\ifx\bbl@KVP@intrapenalty\@nnil\expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@\fi\fi \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini \expandafter\ifx\csname bbl@intsp@languagename\endcsname\@empty\else \\bbl@intraspace\bbl@cl{intsp}\@@\fi\fi % We may override the ini \\bbl@intrapenalty0\@@

⟨∗xetex⟩
\def\BabelStringsDefault{unicode}
\let\xebl@stop\relax
\AddBabelHook{xetex}{encodedcommands}{% 
def\bbl@tempa{#1}% \ifx\bbl@tempa\@empty\XeTeXinputencoding"bytes"\else\XeTeXinputencoding#1\fi\def\xebl@stop{\XeTeXinputencoding"utf8"} % We may override the ini \expandafter\ifx\csname bbl@intsp@languagename\endcsname\@empty\else\expandafter\ifx\bbl@KVP@intraspace\@nnil\expandafter\bbl@intraspace\bbl@KVP@intraspace\@@\fi\ifx\bbl@KVP@intrapenalty\@nnil\expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@\fi\fi \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini \expandafter\ifx\csname bbl@intsp@languagename\endcsname\@empty\else \\bbl@intraspace\bbl@cl{intsp}\@@\fi\fi % We may override the ini \\bbl@intrapenalty0\@@

⟨/Footnote changes⟩
9.2 Layout

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titleps, and geometry. \bbl@startskip and \bbl@endskip are available to package authors. Thanks to the \TeX\ expansion mechanism the following constructs are valid: \adim\bbl@startskip, \advance\bbl@startskip, \bbl@startskip, \bbl@startskip. Consider txtbabel as a shorthand for 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.
9.3 8-bit TeX
Which start just above, because some code is shared with xetex. Now, 8-bit specific stuff.
\begin{verbatim}
\def\bbl@provide@extra#1{%
  \ifx\bbl@encoding@select@off\@empty\else
    \bbl@ifunset{bbl@encoding#1}{% Save last declared
      \@elt##1%\edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}\count@\z@
      \bbl@foreach\bbl@tempe{\def\bbl@tempd{##1}%
        \advance\count@\@ne}%
      \ifnum\count@>\@ne
        \getlocaleproperty{bbl@tempa}{identification/encodings}\ifx\bbl@tempa\relax\let\bbl@tempa\@empty\fi
        \bbl@replace\bbl@tempa{ }{,}\global\bbl@csarg\let{encoding#1}\@empty
        \bbl@xin{,\bbl@tempd,}{,\bbl@tempa,}\ifin@\else% if main encoding included in ini, do nothing
          \let\bbl@tempb\relax
          \bbl@foreach\bbl@tempa{\ifx\bbl@tempb\relax\bbl@xin{,##1,}{,\bbl@tempe,}\ifin@\def\bbl@tempb{##1}\fi}\fi
      \fi}
    \fi}
  \fi}%
\fi}
\end{verbatim}

9.4 LuaTeX
The loader for luatex is based solely on language.dat, which is read on the fly. The code shouldn’t be executed when the format is build, so we check if \AddBabelHook is defined. Then comes a modified version of the loader in hyphen.cfg (without the hyphenmins stuff, which is under the direct control of babel).
The names \<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 runtime.
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 file is read at three places: (1) when plain.def, babel.sty starts, to read the list of available languages from language.dat (for the base option); (2) at hyphen.cfg, to modify some macros; (3) in the middle of plain.def and babel.sty, by babel.def, with the commands and other definitions for luatex (eg, \bbl\patterns).

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

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

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

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

Babel.characters = Babel.characters or {}
Babel.ranges = Babel.ranges or {}
function Babel.hlist_has_bidi(head)
    local has_bidi = false
    local ranges = Babel.ranges
    for item in node.traverse(head) do
        if item.id == node.id'glyph' then
            local itemchar = item.char
            local chardata = Babel.characters[itemchar]
            local dir = chardata and chardata.d or nil
            if not dir then
                for nn, et in ipairs(ranges) do
                    if itemchar < et[1] then
                        break
                    elseif itemchar <= et[2] then
                        dir = et[3]
                        break
                    end
                end
            end
            if dir and (dir == 'al' or dir == 'r') then
                has_bidi = true
            end
        end
    end
end
function Babel.set_chranges_b (script, chrng)
if chrng == '' then return end
texio.write('Replacing ' .. script .. ' script ranges')
Babel.script_blocks[script] = {}
for s, e in string.gmatch(chrng.. ' ', '(.-)%.%.(.-)%s') do
    table.insert(Babel.script_blocks[script], tonumber(s,16), tonumber(e,16))
end
end
function Babel.discard_sublr(str)
if str:find(\[\{\string\indexentry\}] and str:find(\[\{\string\babelsublr\}] then
    str = str:gsub(\[\{\string\babelsublr\%s*(%b{})\]}, function(m) return m:sub(2,-2) end )
return str
end
end
}
This macro adds patterns. Two macros are used to store them: \bbl@patterns@ for the global ones
and \bbl@patterns@<lang> for language ones. We make sure there is a space between words when
multiple commands are used.

\@onlypreamble\babelpatterns
\AtEndOfPackage{%
\newcommand\babelpatterns[2][\@empty]{%
  \ifx\bbl@patterns@\relax
    \let\bbl@patterns@\@empty
  \fi
  \ifx\@empty#1%
    \protected@edef\bbl@patterns@{{\bbl@patterns@ \space#2}}%  
  \else
    \edef\bbl@tempb{{\zap@space#1 \@empty}}%  
    \bbl@for\bbl@tempa\bbl@tempb{%
      \bbl@fixname\bbl@tempa
      \bbl@iflanguage\bbl@tempa{%
        \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
          \@ifundefined{bbl@patterns@\bbl@tempa}{}%  
          \@empty
          {\space}#2}\}%
      }%
    }%
  \fi
}  
}

% TODO - to a lua file
\directlua{
Babel = Babel or {}
Babel.linebreaking = Babel.linebreaking or {}
function Babel.linebreaking.add_before(func, pos)
  tex.print({[noexpand\cname bbl@luahyphenate\endcsname]})
  if pos == nil then
    table.insert(Babel.linebreaking.before, func)
  else
    \\bbl@for\bbl@tempa\bbl@tempb{%
      \bbl@fixname\bbl@tempa
      \bbl@iflanguage\bbl@tempa{%
        \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
          \@ifundefined{bbl@patterns@\bbl@tempa}{}%  
          \@empty
          {\space}#2}\}%
      }%
    }%
else
    table.insert(Babel.linebreaking.before, pos, func)
end

function Babel.linebreaking.add_after(func)
    tex.print(\[
        \noexpand\csname bbl@luahyphenate\endcsname
    \])
    table.insert(Babel.linebreaking.after, func)
end

\begin{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[\thelocaleid].intraspace = %
    \{b = #1, p = #2, m = #3\}
\end{directlua}

\def\bbl@intrapenalty#1\@@{
    \directlua{
        Babel = Babel or {}
        Babel.intrapenalties = Babel.intrapenalties or {}
        Babel.intrapenalties[\csname bbl@sbcp@\languagename\endcsname] = #1
        Babel.locale_props[\thelocaleid].intrapenalty = #1
    }
}
\begin{group}
\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
            if i == node.id'glyph' then
                last_char = item
            elseif i == 7 and item.subtype == 3 and last_char
                and last_char.char > 0x0C99 then
                quad = font.getfont(last_char.font).size
                for lg, rg in pairs(sea_ranges) do
                    if last_char.char > rg[1] and last_char.char < rg[2] then
                        lg = lg:sub(1, 4) ^% Remove trailing number of, eg, Cyril1
                        local intraspace = Babel.intraspaces[lg]
                        local intrapenalty = Babel.intrapenalties[lg]
                        local n
                        if intrapenalty ~= 0 then
                            n = node.new(14, 0) ^% penalty
                            n.penalty = intrapenalty
                        else
                            table.insert(Babel.linebreaking.before, pos, func)
                        end
                    end
                end
            end
        end
    end
    function Babel.linebreaking.add_after(func)
        tex.print(\[
            \noexpand\csname bbl@luahyphenate\endcsname
        \])
        table.insert(Babel.linebreaking.after, func)
end
\end{group}
9.6 **CJK line breaking**

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

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

```latex
\catcode`%=14
\gdef\bbl@cjkintraspacer\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
      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 % )\[ as CL
      if class == 'id' then class = 'I' end
      local br = 0
      if class and last_class and Babel.cjk_breaks[last_class][class] then
        br = Babel.cjk_breaks[last_class][class]
      end
      if br == 1 and props.linebreak == 'c' and
        lang ~= \the\l@nohyphenation\space and
```
last_lang ~\textasciitilde= \textit{nohyphenation} then
local intrapenalty = props.intrapenalty
if intrapenalty ~\textasciitilde= 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
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
  lang.hyphenate(head)
  if Babel.linebreaking.after then
    for k, func in ipairs(Babel.linebreaking.after) do
      func(head)
    end
  end
  if Babel.sea_enabled then
    Babel.sea_disc_to_space(head)
  end
end,
'Babel.hyphenate')
}
\endgroup
\def\bbl@provide@intraspace{%
% \bbl@ifunset{bbl@intsp@\languagename}{}\%
% \ifin@ \let\bbl@cjkintraspace\relax\%
\directlua{\Babel = Babel or {}
Babel.locale_props = Babel.locale_props or {}
Babel.locale_props[the\localeid].linebreak = 'c'

Babel.locale_props[the\localeid].linebreak = 'c'
%\bbl@exp{\bbl@intraspace\bbl@cl{intsp}\@@}%
\ifx\bbl@KVP@intrapenalty\@nnil\fi
\else % sea
\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\fi
\else % sea
\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\fi
\fi
\fi
\ifx\bbl@KVP@intrapenalty\@nnil\else
\expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@\fi
\fi
\ifx\bbl@KVP@intrapenalty\@nnil
\expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
\fi
\fi
\ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
\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,0633,0634,0635,0636,063B,%
063C,063D,063E,063F,0641,0642,0643,0644,0646,0647,0649}\
\begingroup
\catcode`_=11 \catcode`:=11
\gdef\bblar@nofswarn{\gdef\msg_warning:nnx##1##2##3{}}
\endgroup
\gdef\bbl@arabicjust{% TODO. Allow for serveral locales.
\let\bbl@arabicjust\relax
\newattribute\bblar@kashida
\directlua{Babel.attr_kashida = luatexbase.registernumber'\bblar@kashida' }
\bblar@kashida=\z@
\bbl@patchfont{{\bbl@parsejalt}}%
\directlua{
Babel.arabic.elong_map = Babel.arabic.elong_map or {}
Babel.arabic.elong_map[the\localeid] = {}
luatexbase.add_to_callback('post_linebreak_filter',
    Babel.arabic.justify, 'Babel.arabic.justify')
luatexbase.add_to_callback('hpack_filter',
    Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
}}%
\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}%%
\savebothnodeliststomakereplacement. TODO. Save alsowidths to make computations.
Elongated forms. Brute force. No rules at all, yet. The ideal: look at jalt table. And perhaps other tables (falt?, cswh?). What about kaf? And diacritic positioning?

The actual justification (inspired by CHICKENIZE).
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
    % Exclude last line. todo. But-- it discards one-word lines, too!
    % ? Look for glue = 12:15
    if (line.glue_sign == 1 and line.glue_order == 0) then
        elongs = {} % Stores elongated candidates of each line
        k_list = {} % And all letters with kashida
        pos_inline = 0 % Not yet used
        pos_inline = pos_inline + 1 % To find where it is. Not used.
        for n in node.traverse_id(GLYPH, line.head) do
            pos_inline = pos_inline + 1 % To find where it is. Not used.
        end
        % Elongated glyphs
        if elong_map then
            local locale = node.get_attribute(n, LOCALE)
            if elong_map[locale] and elong_map[locale][n.font] and
                elong_map[locale][n.font][n.char] then
                table.insert(elongs, {node = n, locale = locale} )
            end
            node.set_attribute(n.prev, KASHIDA, 0)
            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
            % 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
% If continue, pop the just substituted node from the list:
table.remove(elongs, x)
end

% == Tatwil ==
if #k_list == 0 then goto next_line end

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

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

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
    end
    if cnt > Babel.arabic.kashida_limit then
      node.remove(line.head, n)
    end
  end
else
  cnt = 0
end
end
::next_line::

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

9.8 Common stuff

\AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
\AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
\DisableBabelHook{babel-fontspec}
⟨⟨
Font selection
⟩⟩

9.9 Automatic fonts and ids switching

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

% 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'] = {{0x0980, 0x09FF}},
  ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x182E0, 0x182FF}},
  ['Cyril'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
               {0x1CC0, 0x1D5F}, {0x1E00, 0x1EFF}, {0x1F00, 0x1FFF}}}
{0x2DE0, 0x2DFF}, {0xA640, 0xA69F}),
['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
{0xAB00, 0xAB2F}},
['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
% Don't follow strictly Unicode, which places some Coptic letters in
% the 'Greek and Coptic' block
['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
{0xAB00, 0xAB2F}},
['Hebr'] = {{0x0590, 0x05FF}},
['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
{0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
['Knda'] = {{0x0C80, 0x0CFF}},
['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
{0xA960, 0xA9FF}, {0xAC00, 0xD7AF},
{0x0E00, 0x0EFF}},
['Laoo'] = {{0x0E80, 0x0EFF}},
['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
{0x0180, 0x01EF}, {0x2C60, 0x2C7F},
{0x2F800, 0x2FA1F}},
['Mahj'] = {{0x11150, 0x1117F}},
['Mlym'] = {{0x0D00, 0x0D7F}},
['Mymr'] = {{0x1000, 0x109F}, {0x3000, 0x303F}, {0x3130, 0x318F},
{0x4E00, 0x9FAF}, {0xA960, 0xA9FF}, {0xA0C0, 0x0D7F},
{0x0700, 0x07FF}, {0x0F00, 0x0FF}},
['Nko'] = {{0x0E80, 0x0EFF}},
['Orya'] = {{0x0B00, 0x0B7F}},
['Sinh'] = {{0x1100, 0x11FF}, {0x0700, 0x07FF}, {0x0860, 0x086F},
{0x0B80, 0x0BFF}},
['Telu'] = {{0x0C00, 0x0C7F}},
['Tfng'] = {{0x2D30, 0x2D7F}},
['Thai'] = {{0x0E00, 0x0E7F}},
['Tib'] = {{0x0A00, 0x0A7F}},
['Tnm'] = {{0x0A00, 0x0A4CF}},
['Vai'] = {{0xA500, 0xA63F}},
['Yi'] = {{0xA800, 0xA8FF}, {0x0000, 0x0A4CF}}
}
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
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
  and Babel.locale_props[toloc].script ==
  Babel.locale_props[node.get_attribute(item, LOCALE)].script then
  toloc = nil
end
if toloc then
  if Babel.locale_props[toloc].lg then
    item.lang = Babel.locale_props[toloc].lg
    node.set_attribute(item, LOCALE, toloc)
  end
  if Babel.locale_props[toloc]["\string/..item.font"] then
    item.font = Babel.locale_props[toloc]["\string/..item.font"]
  end
  toloc_save = toloc
else if not inmath and item.id == 7 then % Apply recursively
  item.replace = item.replace and Babel.locale_map(item.replace)
  item.pre = item.pre and Babel.locale_map(item.pre)
  item.post = item.post and Babel.locale_map(item.post)
elseif item.id == node.id'\string/math' then
  inmath = (item.subtype == 0)
end
end
return head
}

The code for \babelcharproperty is straightforward. Just note the modified lua table can be different.
\newcommand\babelcharproperty[1]{%
\count@=#1:\relax
\ifvmode
\expandafter\bbl@chprop
\else
\bbl@error{\string/babelcharproperty\space can be used only in\%
121
vertical mode (preamble or between paragraphs)}
(See the manual for further info)
\fi
\newcommand{\bbl@chprop}[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}}%
\loop
\bbl@cs{chprop@#2}{#3}%
\ifnum\count@<\@tempcnta
\advance\count@\@ne
\repeat}
\def{\bbl@chprop@direction#1}{%
\directlua{%
Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
Babel.characters[\the\count@]["d"] = '#1'
}}%
\let{\bbl@chprop@bc}{\bbl@chprop@direction}
\def{\bbl@chprop@mirror#1}{%
\directlua{%
Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
Babel.characters[\the\count@]["m"] = '\number#1'
}}%
\let{\bbl@chprop@bmg}{\bbl@chprop@mirror}
\def{\bbl@chprop@linebreak#1}{%
\directlua{%
Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
Babel.cjk_characters[\the\count@]["c"] = '#1'
}}%
\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}}
}}%
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.

\begin{verbatim}
Babel.nohyphenation = \the\l@nohyphenation
\end{verbatim}

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, \texttt{pre={1}{1}-} becomes \texttt{function(m) return m[1]..m[1]..'-' end}, where \texttt{m} are the matches returned after applying the pattern. With a mapped capture the functions are similar to \texttt{function(m) return Babel.capt_map(m[1],1) end}, where the last argument identifies the mapping to be applied to \texttt{m[1]}. The way it is carried out is somewhat tricky, but the effect is not dissimilar to Lua load – save the code as string in a \TeX{} macro, and expand this macro at the appropriate place. As \directlua{} does not take into account the current catcode of \texttt{@}, we just avoid this character in macro names (which explains the internal group, too).
\begin{verbatim}
\begingroup
\catcode`\~=12
\catcode`\%=12
\catcode`\&=14
\catcode`\|=12
\gdef\babel@prehyphenation{&%
\@ifnextchar[[\bbl@settransform{0}\bbl@settransform{0}[]]}
\gdef\babel@posthyphenation{&%
\@ifnextchar[[\bbl@settransform{1}\bbl@settransform{1}[]]}
\endgroup
\end{verbatim}
\bbl@transfont@list
\ifnum\count@=\z@
\bbl@exp{\global{\bbl@add{\bbl@transfont@list}
{\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}\&%}
\fi
\bbl@ifunset{\bbl@kv@attribute}\&%\global{\bbl@carg\newattribute{\bbl@kv@attribute}}&%{}\&%\global{\bbl@carg\setattribute{\bbl@kv@attribute}\@ne}\fi
\else
\edef{\bbl@kv@attribute}{\expandafter{\bbl@stripslash}{\bbl@kv@attribute}}&%}
\fi
\directlua{
local lbkr = Babel.linebreaking.replacements[#1]
local u = unicode.utf8
local id, attr, label
if #1 == 0 then
id = \the\csname bbl@id@@#3\endcsname\space
else
id = \the\csname l@#3\endcsname\space
end
ifx{\bbl@kv@attribute}\relax
attr = -1
else
attr = luatexbase.registernumber{\bbl@kv@attribute}'
\fi
{\bbl@kv@label}\&% Same refs:
label = \[==[\bbl@kv@label]==\]
&% Convert pattern:
l_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(patt, '{(%x%x%x%x+)}', function(n)
return u.gsub(u.char(tonumber(n, 16)), '%p', '%%%1')
end)
lbkr[id] = lbkr[id] or {}
table.insert{lbrk[id],
{ label=label, attr=attr, pattern=patt, replace={\babeltempb} }}}
]&%
\endgroup}
\endgroup
\let{\bbl@transfont@list}@empty
\def{\bbl@settransfont}{%
\global{\let{\bbl@settransfont}@relax % Execute only once
\global{\let{\bbl@transfont}@relax % Do nothing if no fonts
\def{\bbl@elt###1###2###3}{%}
\bbl@ifblank{###3}%
{\count@tw}@% Do nothing if no fonts
124
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) }}

9.10 Bidi
As a first step, add a handler for bidi and digits (and potentially other processes) just before luaoftload is applied, which is loaded by default by \LaTeX. Just in case, consider the possibility it has not been loaded.
\def\bbl@activate@preotf{%
6127 \let\bbl@activate@preotf\relax % only once
6128 \directlua{
6129 Babel = Babel or {} 
6130 %
6131 function Babel.pre_otfload_v(head)
6132 if Babel.numbers and Babel.digits_mapped then
6133 head = Babel.numbers(head)
6134 end
6135 if Babel.bidi_enabled then
6136 head = Babel.bidi(head, false, dir)
6137 end
6138 return head
6139 end
6140 %
6141 function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6142 if Babel.numbers and Babel.digits_mapped then
6143 head = Babel.numbers(head)
6144 end
6145 if Babel.bidi_enabled then
6146 head = Babel.bidi(head, false, dir)
6147 end
6148 return head
6149 end
6150 %
6151 luatexbase.add_to_callback('pre_linebreak_filter',
6152 Babel.pre_otfload_v, 
6153 'Babel.pre_otfload_v',
6154 luatexbase.priority_in_callback('pre_linebreak_filter',
6155 'luaotfload.node_processor') or nil)
6156 %
6157 luatexbase.add_to_callback('hpack_filter',
6158 Babel.pre_otfload_h, 
6159 'Babel.pre_otfload_h',
6160 luatexbase.priority_in_callback('hpack_filter',
6161 'luaotfload.node_processor') or nil)
6162 })

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

6163 \breakafterdirmode=1
6164 \ifnum\bbl@bidimode=1 % Any bidi= except default=1
6165 \let\bbl@beforeforeign\leavevmode
6166 \AtEndOfPackage{\EnableBabelHook{babelf-bidi}}
6167 \RequirePackage{luatexbase}
6168 \bbl@activate@preotf
6169 \directlua{
6170 require('babel-data-bidi.lua')
6171 \ifcase\expandafter@gobbletwo\the\bbl@bidimode\or
6172 require('babel-bidi-basic.lua')
6173 \or
6174 require('babel-bidi-basic-r.lua')
6175 \fi}
6176 \newattribute\bbl@attr@dir
6177 \directlua{ Babel.attr_dir = luatexbase.registernumber'\bbl@attr@dir' }
6178 \bbl@exp\{'output\{bodydir\pagedir\the\output\}'
6179 \fi
6180 \chardef\bbl@thetextdir\z@  
6181 \chardef\bbl@thepardir\z@
6182 \def\bbl@getluadir#1{%
6183 \directlua{
6184 if tex.#1dir == 'TLT' then
6185 tex.sprint('0')
6186 end

9.11 Layout

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

\directlua{function Babel.math_box_dir(head)
  if not (token.get_macro('bbl@insidemath') == '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
  return head
end
luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir, "Babel.math_box_dir", 0)}
reliable.

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

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

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

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, tabular seems to work (at least in simple cases) with array, tabularx, hhline, colortbl, longtable, booktabs, etc. However, dcolumn still fails.
elseif item.id == node.id 'math' then
    inmath = (item.subtype == 0)
end
end

return head
Very likely the output routine must be patched in a quite general way to make sure the \bodydir is set to \pagedir. Note outside output they can be different (and often are). For the moment, two ad hoc changes.

\AtBeginDocument{%
\IfPackageloaded{multicol}%
{\toks@\expandafter{\multi@column@out}%
  \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
}%
\AtBeginDocument{\ifpackageloaded{paracol}%
  {\edef\pcol@output{\bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
  {}%}
\fi
\ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout

\begin{verbatim}
\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.
\end{verbatim}
\let\bbl@NL@tabular@tabular
\AtBeginDocument{% 
\if\bbl@NL@tabular@tabular\else 
\bbl@exp{\ifin@{\bbl@nextfake}{[@tabular]}% 
\ifin\else 
\bbl@replace@tabular{$}{\bbl@nextfake$}% 
\fi 
\let\bbl@NL@tabular@tabular
\fi}%
\IfBabelLayout{lists}{
\let\bbl@OL@list\list 
\bbl@sreplace\list{\parshape}{\bbl@listparshape} 
\let\bbl@NL@list\list 
\def\bbl@listparshape#1#2#3{% 
\parshape #1 #2 #3 % 
\ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else 
\shapemode\tw@ 
\fi}

\IfBabelLayout{graphics}{
\let\bbl@pictresetdir\relax 
\def\bbl@pictsetdir#1{% 
\ifcase\bbl@thetextdir 
\let\bbl@pictresetdir\relax 
\else 
\ifcase#1\bodydir TLT % Remember this sets the inner boxes 
\or\textdir TLT 
\else\bodydir TLT \textdir TLT 
\fi 
\fi 
% \text|par)dir required in pgf: 
\def\bbl@pictsetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}% 
\fi}
\AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
\directlua{ 
Babel.get_picture_dir = true 
Babel.picture_has_bidi = 0 
% function Babel.picture_dir (head) 
if not Babel.get_picture_dir then return head end 
if Babel.hlist_has_bidi(head) then 
Babel.picture_has_bidi = 1 
end 
return head 
end 
\luatexbase.add_to_callback("hpack_filter", Babel.picture_dir, 
"Babel.picture_dir")
}%
\AtBeginDocument{% 
\def\LS@rot{% 
\setbox\@outputbox\vbox{% 
\hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}}% 
\long\def\put(#1,#2)#3{% 
\@killglue 
% Try: 
\ifx\bbl@pictresetdir\relax 
\def\bbl@tempc{0}% 
\else 
\directlua{ 
Babel.get_picture_dir = true 
Babel.picture_has_bidi = 0 
}% 
\setbox\z@\hbox{\tempc{}}

133
Implicitly reverses sectioning labels in bidi=basic-r, because the full stop is not in contact with L numbers any more. I think there must be a better way. Assumes bidi=basic, but there are some additional readjustments for bidi=default.
Some \LaTeX{} macros use internally the math mode for text formatting. They have very little in common and are grouped here, as a single option.

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: \texttt{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.); \texttt{fetch\_word} fetches a series of glyphs and discretionaries, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck).

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

\section{Lua: transforms}

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: \texttt{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.); \texttt{fetch\_word} fetches a series of glyphs and discretionaries, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck).

\texttt{post\_hyphenate\_replace} is the callback applied after \texttt{lang\_hyphenate}. This means the automatic hyphenation points are known. As empty captures return a byte position (as explained in the luatex manual), we must convert it to a utf8 position. With first, the last byte can be the leading byte in a utf8 sequence, so we just remove it and add 1 to the resulting length. With \texttt{last} we must take into account the capture position points to the next character. Here \texttt{word\_head} points to the starting node of the text to be matched.
Babel.fetch_subtext = {}
Babel.ignore_pre_char = function(node)
  return (node.lang == Babel.nohyphenation)
end

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

    while item do
        if item.id == 11 then
            inmath = (item.subtype == 0)
        end

        if inmath then
            -- pass
        elseif item.id == 29 then
            if item.lang == lang or lang == nil then
                if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
                    lang = lang or item.lang
                    word_string = word_string .. unicode.utf8.char(item.char)
                    word_nodes[#word_nodes+1] = item
                end
            else
                break
            end
        elseif item.id == 7 and item.subtype == 2 then
            word_string = word_string .. '='
            word_nodes[#word_nodes+1] = item
        elseif item.id == 7 and item.subtype == 3 then
            word_string = word_string .. '|'..'
            word_nodes[#word_nodes+1] = item
        else
            if word_string == '' then
                -- pass
            elseif (item.id == 12 and item.subtype == 13) then
                break
            else
                word_string = word_string .. Babel.us_char
                word_nodes[#word_nodes+1] = item -- Will be ignored
            end
        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)

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
                -- subsubstrings.
                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
            end
        end
    end

138
-- 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)
    local ss = ''
    for itt in node.traverse(head) do
      if itt.id == 29 then
        ss = ss .. unicode.utf8.char(itt.char)
      else
        ss = ss .. '{' .. itt.id .. '}'
      end
    end
    print('*****************', ss)
  end

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

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

  if crep then
    step = crep.step or 0
  end
if (not enabled) or (crep and next(crep) == nil) then -- = {}
    last_match = save_last -- Optimization
    goto next
elseif crep == nil or crep.remove then
    node.remove(head, item)
table.remove(w_nodes, sc)
w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
sc = sc - 1 -- Nothing has been inserted.
last_match = utf8.offset(w, sc+1+step)
goto next
elseif crep == nil or crep.remove then
    node.remove(head, item)
table.remove(w_nodes, sc)
w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
sc = sc - 1 -- Nothing has been inserted.
last_match = utf8.offset(w, sc+1+step)
goto next
elseif crep and crep.kashida then -- Experimental
    node.set_attribute(item, Babel.attr_kashida, crep.kashida)
    last_match = utf8.offset(w, sc+1+step)
goto next
elseif crep and crep.string then
    local str = crep.string(matches)
    if str == '' then -- Gather with nil
        node.remove(head, item)
table.remove(w_nodes, sc)
w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
sc = sc - 1 -- Nothing has been inserted.
else
    local loop_first = true
    for s in string.utfvalues(str) do
        d = node.copy(item_base)
d.char = s
        if loop_first then
            loop_first = false
            head, new = node.insert_before(head, item, d)
            if sc == 1 then
                word_head = head
            end
            w_nodes[sc] = d
            w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
        else
            sc = sc + 1
            head, new = node.insert_before(head, item, d)
table.insert(w_nodes, sc, new)
w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
        end
        if Babel.debug then
            print('.....', 'str')
            Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
        end
    end -- for
end -- if ''
last_match = utf8.offset(w, sc+1+step)
goto next
elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
    d = node.new(7, 3) -- (disc, regular)
d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
d.post = Babel.str_to_nodes(crep.post, matches, item_base)
d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
d.attr = item_base.attr
if crep.pre == nil then -- TeXbook p96
d.penalty = crep.penalty or tex.hyphenpenalty
else
d.penalty = crep.penalty or tex.exhyphenpenalty
end
placeholder = '|' head, new = node.insert_before(head, item, d)
elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
-- ERROR
elseif crep and crep.penalty then
d = node.new(14, 0) -- (penalty, userpenalty)
d.attr = item_base.attr
d.penalty = crep.penalty
head, new = node.insert_before(head, item, d)
elseif crep and crep.space then
-- 655360 = 10 pt = 10 * 65536 sp
local quad = font.getfont(item_base.font).size or 655360
node.setglue(d, crep.space[1] * quad,
crep.space[2] * quad,
crep.space[3] * quad)
if mode == 0 then
placeholder = ' ' end
head, new = node.insert_before(head, item, d)
elseif crep and crep.spacefactor then
d = node.new(12, 13) -- (glue, spaceskip)
local base_font = font.getfont(item_base.font)
node.setglue(d,
crep.spacefactor[1] * base_font.parameters['space'],
crep.spacefactor[2] * base_font.parameters['space_stretch'],
crep.spacefactor[3] * base_font.parameters['space_shrink'])
if mode == 0 then
placeholder = ' ' end
head, new = node.insert_before(head, item, d)
elseif mode == 0 and crep and crep.space then
-- ERROR
end -- ie replacement cases
-- Shared by disc, space and penalty.
if sc == 1 then
word_head = head end
if crep.insert then
w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
table.insert(w_nodes, sc, new)
last = last + 1
else
w_nodes[sc] = d
node.remove(head, item)
w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
end
last_match = utf8.offset(w, sc+1+step)
::next::
end -- for each replacement

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

end -- for match

end -- for patterns
::next::
word_head = nw
end -- for substring
return head
end

-- This table stores capture maps, numbered consecutively
Babel.capture_maps = {}
-- The following functions belong to the next macro
function Babel.capture_func(key, cap)
    local ret = "[[[.. cap.gsub('[[[0-9]]]', '', ..m[1] .. '..]]"]
    local cnt
    local u = unicode.utf8
    ret, cnt = ret:gsub('{([0-9])|([^[|]+)|(.-)}', Babel.capture_func_map)
    if cnt == 0 then
        ret = u.gsub(ret, '{(%x%x%x%x+)}', function (n)
            return u.char(tonumber(n, 16))
        end)
    end
    return key .. [[=function(m) return ]] .. ret .. [[ end]]
end

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

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

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

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

9.13 Lua: Auto bidi with basic and basic-r

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

```lua
[0x25]={d='et'},
[0x26]={d='on'},
[0x27]={d='on'},
```
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
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'
```

144
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
            dir = dir or 'l'
        end
        if inmath then dir = ('TRT' == tex.mathdir) and 'r' or 'l' end
    end
    if new_dir then
        attr_dir = 0
        for at in node.traverse(item.attr) do
            if at.number == Babel.attr_dir then
                attr_dir = at.value & 0x3
            end
        end
        if attr_dir == 1 then
            strong = 'r'
        elseif attr_dir == 2 then
            strong = 'al'
        else
            strong = 'l'
        end
        strong_lr = (strong == 'l') and 'l' or 'r'
        outer = strong_lr
        new_dir = false
    end
end
if dir == 'nsm' then dir = strong end  -- W1

Numbers. The dual <al>/<r> system for R is somewhat cumbersome.
7278  dir_real = dir  -- We need dir_real to set strong below
7279  if dir == 'al' then dir = 'r' end  -- W3

By W2, there are no <en> <et> <es> if strong == <al>, only <an>. Therefore, there are not <et en> nor <en et>, W5 can be ignored, and W6 applied:
7280  if strong == 'al' then
7281    if dir == 'en' then dir = 'an' end  -- W2
7282    if dir == 'et' or dir == 'es' then dir = 'on' end  -- W6
7283    strong_lr = 'r'  -- W3
7284  end

Once finished the basic setup for glyphs, consider the two other cases: dir node and the rest.
7285  elseif item.id == node.id'dir' and not inmath then
7286    new_dir = true
7287    dir = nil
7288  elseif item.id == node.id'math' then
7289    inmath = (item.subtype == 0)
7290  else
7291    dir = nil  -- Not a char
7292  end

Numbers in R mode. A sequence of <en>, <et>, <an>, <es> and <cs> is typeset (with some rules) in L mode. We store the starting and ending points, and only when anything different is found (including nil, ie, a non-char), the textdir is set. This means you cannot insert, say, a whatsit, but this is what I would expect (with luacolor you may colorize some digits). Anyway, this behavior could be changed with a switch in the future. Note in the first branch only <an> is relevant if <al>.
7293  if dir == 'en' or dir == 'an' or dir == 'et' then
7294    if dir ~= 'et' then
7295      type_n = dir
7296    end
7297    first_n = first_n or item
7298    last_n = last_es or item
7299    last_es = nil
7300  elseif dir == 'es' and last_n then  -- W3+W6
7301    last_es = item
7302  elseif dir == 'cs' then  -- it's right - do nothing
7303    elseif first_n then  -- & if dir = any but en, et, an, es, cs, inc nil
7304      if strong_lr == 'r' and type_n ~= '' then
7305        dir_mark(head, first_n, last_n, 'r')
7306      elseif strong_lr == 'l' and first_d and type_n == 'an' then
7307        dir_mark(head, first_n, last_n, 'r')
7308      elseif strong_lr == 'l' and first_d and type_n == 'a' then
7309        dir_mark(head, first_d, last_d, outer)
7310      elseif strong_lr == 'l' and type_n ~= '' then
7311        last_d = last_n
7312      end
7313      type_n = ''
7314      first_n, last_n = nil, nil
7315    end

R text in L, or L text in R. Order of dir_mark's are relevant: d goes outside n, and therefore it's emitted after. See dir_mark to understand why (but is the nesting actually necessary or is a flat dir structure enough?). Only L, R (and AL) chars are taken into account – everything else, including spaces, whatsits, etc., are ignored:
7316  if dir == 'l' or dir == 'r' then
7317    if dir ~= outer then
7318      first_d = first_d or item
7319      last_d = item
7320    elseif first_d and dir ~= strong_lr then
7321      dir_mark(head, first_d, last_d, outer)
7322    first_d, last_d = nil, nil
7323  end
7324  end

146
Mirroring. Each chunk of text in a certain language is considered a “closed” sequence. If `<r` on `<r` and `<l` on `<l`, it’s clearly `<r` and `<l`, respctly, but with other combinations depends on outer. From all these, we select only those resolving `<on` → `<r`. At the beginning (when last_lr is nil) of an R text, they are mirrored directly. 

TODO - numbers in R mode are processed. It doesn’t hurt, but should not be done.

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).

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

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

And here the Lua code for bidi=basic:

Babel = Babel or {}

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

Babel.fontmap = Babel.fontmap or {}

Babel.fontmap[0] = {} -- l

Babel.fontmap[1] = {} -- r

Babel.bidi_enabled = true
Babel.mirroring_enabled = true
require('babel-data-bidi.lua')
local characters = Babel.characters
local ranges = Babel.ranges
local DIR = node.id('dir')
local GLYPH = node.id('glyph')
local function insert_implicit(head, state, outer)
  local new_state = state
  if state.sim and state.eim and state.sim ~= state.eim then
    dir = ((outer == 'R') and 'TLT' or 'TRT') -- ie, reverse
    local d = node.new(DIR)
    d.dir = '+' .. dir
    node.insert_before(head, state.sim, d)
    local d = node.new(DIR)
    d.dir = '-' .. dir
    node.insert_after(head, state.eim, d)
  end
  new_state.sim, new_state.eim = nil, nil
  return head, new_state
end
local function insert_numeric(head, state)
  local new_state = state
  if state.san and state.ean and state.san ~= state.ean then
    local d = node.new(DIR)
    d.dir = '+TLT'
    _, new = node.insert_before(head, state.san, d)
    if state.san == state.sim then state.sim = new end
    local d = node.new(DIR)
    d.dir = '-TLT'
    _, new = node.insert_after(head, state.ean, d)
    if state.ean == state.eim then state.eim = new end
  end
  new_state.san, new_state.ean = nil, nil
  return head, new_state
end
local function Babel.bidi(head, ispar, hdir)
  local d -- d is used mainly for computations in a loop
  local prev_d = ''
  local new_d = false
  local nodes = {}
  local outer_first = nil
  local inmath = false
  local glue_d = nil
  local glue_i = nil
  local has_en = false
  local first_et = nil
  -- TODO - \hbox with an explicit dir can lead to wrong results
  -- <R \hbox dir TLT(<R>)> and <L \hbox dir TRT(<L>)>. A small attempt
  -- was made to improve the situation, but the problem is the 3-dir
  -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
  -- well.
  function Babel.bidi(head, ispar, hdir)
local 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')
else
  -- Or error? Shouldn't happen
  save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
else
  -- Or error? Shouldn't happen
  save_outer = ('TRT' == hdir) and 'r' or 'l'
end
local outer = save_outer
local last = outer
-- 'al' is only taken into account in the first, current loop
if save_outer == 'al' then save_outer = 'r' end

local fontmap = Babel.fontmap
for item in node.traverse(head) do
  -- In what follows, #node is the last (previous) node, because the
  -- current one is not added until we start processing the neutrals.
  -- three cases: glyph, dir, otherwise
  if item.id == GLYPH
     or (item.id == 7 and item.subtype == 2) then
    local d_font = nil
    local item_r
    if item.id == 7 and item.subtype == 2 then
      item_r = item.replace -- automatic discs have just 1 glyph
    else
      item_r = item
    end
    local chardata = characters[item_r.char]
    d = chardata and chardata.d or nil
    if not d or d == 'nsm' then
      for nn, et in ipairs(ranges) do
        if item_r.char < et[1] then
          break
        elseif item_r.char <= et[2] then
          if not d then d = et[3]
          elseif d == 'nsm' then d_font = et[3]
        end
      end
      break
    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 == 'nsn' and 0) or
    (d_font == 'r' and 1) or
    (d_font == 'al' and 2) or
    (d_font == 'an' and 2) or nil
if d_font and fontmap and fontmap[d_font][item_r.font] then
    item_r.font = fontmap[d_font][item_r.font]
end

if new_d then
    table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
    if inmath then
        attr_d = 0
    else
        attr_d = node.get_attribute(item, ATDIR)
        attr_d = attr_d & 0x3
    end
    if attr_d == 1 then
        outer_first = 'r'
        last = 'r'
    elseif attr_d == 2 then
        outer_first = 'r'
        last = 'al'
    else
        outer_first = 'l'
        last = 'l'
    end
    outer = last
    has_en = false
    first_et = nil
    new_d = false
end

if glue_d then
    if (d == 'l' and 'l' or 'r') ~= glue_d then
        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
elseif item.id == node.id'math' then
    inmath = (item.subtype == 0)
else
    has_hyperlink = true
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' -- W5
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)
elself d == 'en' then
    has_en = true
    first_et = first_et or (#nodes + 1)
else if first_et then -- d may be nil here!
    if has_en then
        if last == 'l' then
            temp = 'l' -- W7
        else
            temp = 'en' -- W5
        end
    else
        temp = 'on' -- W6
    end
    for e = first_et, #nodes do
        if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
    end
    first_et = nil
    has_en = false
end
-- Force mathdir in math if ON (currently works as expected only
-- with 'l')
if inmath and d == 'on' then
    d = ('TRT' == tex.mathdir) and 'r' or 'l'
end
if d then
    if d == 'al' then
        d = 'r'
        last = 'al'
elseif d == 'l' or d == 'r' then
        last = d
    end
    prev_d = d
    table.insert(nodes, {item, d, outer_first})
end
outer_first = nil
end

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

-- 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
    elseif
      temp = outer
    end
    for r = first_on, q - 1 do
      nodes[r][2] = temp
      item = nodes[r][1] -- MIRRORING
      if Babel.mirroring_enabled and item.id == GLYPH
        and temp == 'r' and characters[item.char] then
        local font_mode = ''
        if item.font > 0 and font.fonts[item.font].properties then
          font_mode = font.fonts[item.font].properties.mode
        end
        if font_mode ~= 'harf' and font_mode ~= 'plug' then
          item.char = characters[item.char].m or item.char
        end
      end
      end
    end
  end
  temp = outer
end
for r = first_on, q - 1 do
  nodes[r][2] = temp
  item = nodes[r][1] -- MIRRORING
  if Babel.mirroring_enabled and item.id == GLYPH
    and temp == 'r' and characters[item.char] then
    local font_mode = ''
    if item.font > 0 and font.fonts[item.font].properties then
      font_mode = font.fonts[item.font].properties.mode
    end
    if font_mode ~= 'harf' and font_mode ~= 'plug' then
      item.char = characters[item.char].m or item.char
    end
  end
end
first_on = nil

152
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 == 'ns' 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
      state.eim = item
    elseif d == 'l' and state.sim and state.has_r then
      head, state = insert_implicit(head, state, outer)
    elseif d == 'l' then
      state.sim, state.eim, state.has_r = nil, nil, false
    end
  else
    if d == 'an' or d == 'l' then
      if nodes[q][3] then -- nil except after an explicit dir
        state.sim = item -- so we move sim 'inside' the group
      else
        state.sim = state.sim or item
      end
      state.eim = item
    elseif d == 'r' and state.sim then
      head, state = insert_implicit(head, state, outer)
    elseif d == 'r' then
      state.sim, state.eim = nil, nil
    end
  end
if isdir then
  last = d -- Don't search back - best save now
else if d == 'on' and state.san then
  state.san = state.san or item
  state.ean = item
end
10 Data for CJK

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

```
[0x08021] = {c='ex'},
[0x08024] = {c='pr'},
[0x08025] = {c='po'},
[0x08028] = {c='op'},
[0x08029] = {c='cp'},
[0x0802B] = {c='pr'},
```

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

11 The ‘nil’ language

This ‘language’ does nothing, except setting the hyphenation patterns to nohyphenation.
For this language currently no special definitions are needed or available.
The macro \LdfInit takes care of preventing that this file is loaded more than once, checking the
category code of the @ sign, etc.
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.

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

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

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

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.

12 Calendars

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

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

```
⟨⟨∗ Compute Julian day ⟩⟩ ≡ 155
```
The code for the Civil calendar is based on it, too.

\ExplSyntaxOn
\ExplSyntaxOff

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

Since the main aim is to provide a suitable \today, and maybe some close dates, data just covers Hijri ∼1435/∼2014 (Gregorian ∼2014/∼2038).

12.1 Islamic

The Civil calendar.

The Ummal-Quracalendar,usedmainlyinSaudiArabia,isbasedonmoment-hijri,byAbdullahAlsigar(licenseMIT).
This is basically the set of macros written by Michail Rozman in 1991, with corrections and adaptions by Rama Porrat, Misha, Dan Haran and Boris Lavva. This must be eventually replaced by computations with l3fp. An explanation of what's going on can be found in hebcal.sty

12.2 Hebrew

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

157
\newif\ifbbl@gregleap
\def\bbl@ifgregleap#1{\ifbbl@gregleaptrue\else\fi}
\def\bbl@ifgregleap#1{\ifbbl@gregleapfalse\else\fi}
\def\bbl@gregdayspriormonths#1#2#3{\ifcase#10or0or31or59or90or120or151or181or212or243or273or304or334\fi\bbl@ifgregleap{#2}\ifnum#1>2\advance#3by1\fi\global\bbl@cntcommon=#3}
\def\bbl@gregdaysprioryears#1#2{\countdef\tmpc=4\countdef\tmpb=2\tmpb=#1\relax\advance\tmpbby-1\tmpc=\tmpb\multiply\tmpcby365\ifnum\tmpc=365\#2=\tmpc\global\bbl@cntcommon=#2\relax\fi\global\bbl@cntcommon=#2}
\def\bbl@absfromgreg#1#2#3#4{\countdef\tmpd=0\global\bbl@cntcommon=\#3\#3=\bbl@cntcommon\def\bbl@gregdayspriormonths#1#2%{\countdef\tmpc=4\countdef\tmpb=2\tmpb=\#1\relax\tmpc=\tmpb\divide\tmpcby4\advance\#2by\tmpc\tmpc=\tmpb\divide\tmpcby100\advance\#2by\tmpc\tmpc=\tmpb\divide\tmpcby400\advance\#2by\tmpc\global\bbl@cntcommon=\#2\relax\#2=\bbl@cntcommon\def\bbl@gregdaysprioryears#1%{\global\bbl@cntcommon=\#2\relax}\#2=\bbl@cntcommon\countdef\tmpd=0\global\bbl@cntcommon=\#3\#3=\bbl@cntcommon\def\bbl@gregdayspriormonths#2%{\#3=\tmpd\global\bbl@cntcommon=\#2\relax}\#2=\bbl@cntcommon\def\bbl@gregdaysprioryears#3%{\global\bbl@cntcommon=\#4\relax\#4=\bbl@cntcommon}\def\bbl@absfromgreg#1#2#3#4%{\countdef\tmpd=0\#4=\#1\relax\bbl@gregdayspriormonths#2%{\#3=\tmpd}\global\bbl@cntcommon=\#4\relax\#4=\bbl@cntcommon}\def\bbl@absfromgreg#1#2%{\global\bbl@cntcommon=\#2\relax}\global\bbl@cntcommon=\#3\def\bbl@gregdaysprioryears#2%{\global\bbl@cntcommon=\#4\relax}\global\bbl@cntcommon=\#3%\def\bbl@gregdayspriormonths#1#2%{\global\bbl@cntcommon=\#3\def\bbl@gregdaysprioryears#3%{\global\bbl@cntcommon=\#4\relax}}\global\bbl@cntcommon=\#2%\def\bbl@gregdayspriormonths#1#2%{\global\bbl@cntcommon=\#3%\def\bbl@gregdaysprioryears#3%{\global\bbl@cntcommon=\#4%}}\global\bbl@cntcommon=\#3}}
\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{\percent
\countdef \tmpa=0
\countdef \tmpb=1
\countdef \tmpc=2
\tmpa=#1\relax
\advance \tmpa by -1
#2=\tmpa
\divide #2 by 19
\multiply #2 by 235
\bbl@remainder{\tmpa}{19}\{\tmpb\%
\tmpc=\tmpb
\multiply \tmpb by 12
\advance #2 by \tmpb
\multiply \tmpc by 7
\advance \tmpc by 1
\divide \tmpc by 19
\advance #2 by \tmpc
\global \bbl@cntcommon=#2\%
#2=\bbl@cntcommon}
\def \bbl@hebrelapseddays#1#2{\percent
\countdef \tmpa=0
\countdef \tmpb=1
\countdef \tmpc=2
\bbl@hebrelapsedmonths{#1}{#2}\%
\tmpa=years\{19\}-years this cycle
\tmpc=\tmpb
\divide \tmpc by 12
\advance \tmpc by 7
\divide \tmpc by 19
\divide \tmpc by 19
\global \bbl@cntcommon=\{2\%
#2=\bbl@cntcommon}
\def \bbl@hebrelapseddays#1#2{\percent
\countdef \tmpa=0
\countdef \tmpb=1
\countdef \tmpc=2
\bbl@hebrelapsedmonths(#1){#2}\%
\tmpa=#2\relax
\multiply \tmpa by 13753
\divide \tmpa by 5604
\bbl@hebrelapsedmonths\{25920\}\{\tmpc\%
\tmpc == ConjunctionParts
\divide \tmpc 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
    \ifnum \tmpc < 16789
      \else
      \ifnum \tmpc = \{2\%
        \bbl@checkleaphebryear\{#1\}% of a common year
        \ifbbl@hebrleap
          \else
          \advance #2 by 1
        \fi
        \fi
      \fi
    \fi
  \fi
\else
  \ifnum \tmpc < 16789
    \else
    \ifnum \tmpc = \{1\%
      \bbl@checkleaphebryear\{#1\}% at the end of leap year
      \ifbbl@hebrleap
        \advance #2 by 1
      \fi
      \fi
    \fi
  \fi
\else
  \advance #2 by 1
\fi
\bbl@remainder{#2}{7}{\tmpa}\
\ifnum \tmpa=0
\advance #2 by 1
\else
\fi
\ifnum \tmpa=3
\advance #2 by 1
\else
\fi
\ifnum \tmpa=5
\advance #2 by 1
\fi
\fi
\global\bbl@cntcommon=#2\relax}\
#2=\bbl@cntcommon}
def\bbl@daysinhebryear#1#2{\
{\countdef\tmpe=12
\bbl@hebrelapseddays{#1}{\tmpe}\
\advance #1 by 1
\bbl@hebrelapseddays{#1}{#2}\
\advance #2 by -\tmpe
\global\bbl@cntcommon=#2}\
#2=\bbl@cntcommon}
def\bbl@hebrdayspriormonths#1#2#3{\
{\countdef\tmpf= 14
#3=\ifcase #1 \relax
  0 \ or
  0 \ or
  30 \ or
  59 \ or
  89 \ or
  118 \ or
  148 \ or
  148 \ or
  177 \ or
  207 \ or
  236 \ or
  266 \ or
  295 \ or
  325 \ or
  400
\fi
\fi
\bbl@checkleaphebryear[#2]\
\ifbbl@hebrleap
\ifnum #1 > 6
\advance #3 by 30
\fi
\fi
\ifnum #1 > 3
\ifnum \tmpf=353
\advance #3 by -1
\fi
\ifnum \tmpf=383
\advance #3 by -1
\fi
\fi
\ifnum #1 > 2
\ifnum \tmpf=355
\advance #3 by 1
\fi
\ifnum \tmpf=385
\advance #3 by 1
\fi
\fi
There is an algorithm written in TeX by Jabri, Abolhassani, Pournader and Esfahbod, created for the first versions of the Farsi TeX system (no longer available), but the original license is GPL, so its use with LPPL is problematic. The code here follows loosely that by John Walker, which is free and accurate, but sadly very complex, so the relevant data for the years 2013-2050 have been pre-calculated and stored. Actually, all we need is the first day (either March 20 or March 21).

12.3 Persian

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

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

12.5 Buddhist

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

13.1 Not renaming *hyphen*.tex

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

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

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

The files *bplain*.tex and *blplain*.tex can be used as replacement wrappers around plain.tex and lplain.tex to achieve the desired effect, based on the babel package. If you load each of them with 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}
\cf{\catcode\{=1 \catcode\}=2 \catcode\#=6}
\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 not to waste multi-letter control sequence on this in the format).

\begin{verbatim}
\openin 0 hyphen.cfg
\ifeof0
\let\a\input
Then \input is defined to forget about its argument and load *hyphen*.cfg instead. Once that's done the original meaning of \input can be restored and the definition of \a can be forgotten.
\def\input #1 {\let\input\a
\a*hyphen*.cfg
\let\a\undefined
}
\fi
\end{verbatim}

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

\begin{verbatim}
\(bplain)\a*plain*.tex
\end{verbatim}

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.

\begin{verbatim}
\(bplain)\def\fmtname{babel-plain}
\end{verbatim}
13.2 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 \texttt{\babeloptionstrings} and \texttt{\babeloptionmath} are provided, which can be defined before loading babel. \texttt{\BabelModifiers} can be set too (but not sure it works).

13.3 General tools

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

\begin{verbatim}
\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
\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
\let\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	oks@}}}
\def\@namedef#1{\expandafter\def\csname #1\endcsname}
\def\@nameuse#1{\csname #1\endcsname}
\def\@ifundefined#1{\expandafter\ifx\csname#1\endcsname\relax\expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
\def\@expandtwoargs#1#2#3{\edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
\def\zap@space#1 #2{#1}
\end{verbatim}

165
LATEX has the command \@onlypreamble which adds commands to a list of commands that are no longer needed after \begin{document}.

\begin{verbatim}
\def\begindocument{\@begindocumenthook}
\ifx\@begindocumenthook\@undefined
  \def\@begindocumenthook{}
\fi
\@onlypreamble\@begindocumenthook
\def\AtBeginDocument{\g@addto@macro\@begindocumenthook}
\end{verbatim}

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

\begin{verbatim}
\def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
\end{verbatim}

\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 \texttt{\ifx}. The same trick is applied below.

\begin{verbatim}
\catcode`\&=3
\ifx&if@filesw\@undefined
  \expandafter\let\csname if@filesw\expandafter\endcsname\csname iffalse\endcsname
\fi
\catcode`\&=4
\end{verbatim}
Mimicking \TeX's commands to define control sequences.

\def\newcommand\@star@or@long\newcommand
\@testopt{\@newcommand\#1}{0}
\def\@newcommand\#1\[#2\]{\@ifnextchar\[\@xargdef\#1\[#2\]}{\@argdef\#1\[#2\]}
\long\def\@argdef\#1\[#2\]\#3{\@yargdef\#1\@ne{\#2}{\#3}}
\long\def\@xargdef\#1\[#2\]\[#3\]\#4{\expandafter\def\expandafter\#1\expandafter{\expandafter\@protected@testopt\expandafter\#1\csname\string\#1\endcsname{\#3}}\expandafter\@yargdef\csname\string\#1\endcsname{\#2}{\#4}}
\def\providecommand\@star@or@long\providecommand
\begingroup\escapechar\m@ne\xdef\@gtempa{{\string\#1}}\endgroup\expandafter\@ifundefined\@gtempa{\def\reserved@a{\newcommand\reserved@a}}{\let\reserved@a\relax\def\reserved@a{\newcommand\reserved@a}}\reserved@a
\def\DeclareRobustCommand\@star@or@long\declared@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\#1\fi\noexpand\protect\expandafter\noexpand\csname\expandafter\@gobble\string\#1 \endcsname}\expandafter\newcommand\csname\expandafter\@gobble\string\#1 \endcsname}
\catcode`&=\z@ % Trick to hide conditionals\edef\@x@protect\#1&fi\#2\#3{&fi\protect\#1}

The following little macro \in@ is taken from \latex\ .\tex; 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.

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

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).

\begin{verbatim}
\def@ifpackagewith#1#2#3#4{#3}
\end{verbatim}

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.

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

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

\begin{verbatim}
\ifx\@tempcnta\@undefined
  \csname newcount\endcsname\@tempcnta\relax
\fi
\ifx\@tempcntb\@undefined
  \csname newcount\endcsname\@tempcntb\relax
\fi
\if\bye\@undefined
  \advance\count10 by -2\relax
\fi
\if\@ifnextchar\@undefined
  \def\@ifnextchar#1#2#3{\let\reserved@d=#1\let\reserved@a{#2}\let\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\reserved@c}
\def\:{\let\@sptoken=} \: % this makes \@sptoken a space token
\expandafter\def:\ {uturelet\@let@token\@ifnch}
\end{verbatim}

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

\begin{verbatim}
\if\bye\@undefined
  \advance\count10 by -2\relax
\fi
\if\@ifnextchar\@undefined
  \def\@ifnextchar#1#2#3{\let\reserved@d=#1\def\reserved@b{#2}\futurelet\@let@token\@ifnch}
\def\@ifnch{\ifx\@let@token\@sptoken\let\reserved@c\@xifnch\else\ifx\@let@token\reserved@d\let\reserved@c\reserved@b\else\let\reserved@c\reserved@b\fi\fi\reserved@c}
\end{verbatim}
13.4 Encoding related macros

Code from loutenc.dtx, adapted for use in the plain TeX environment.

\def\DeclareTextCommand{\@dec@text@cmd\providecommand}
\def\ProvideTextCommand{\@dec@text@cmd\providecommand}
\def\DeclareTextSymbol#1#2#3{\@dec@text@cmd\chardef#1{#2}#3\relax}
\def\@dec@text@cmd#1#2#3{\expandafter\def\expandafter#2\expandafter{\csname#3-cmd\expandafter\endcsname#2\csname#3\string#2\endcsname}\%}
\let\@ifdefinable\@rc@ifdefinable
\expandafter\@current@cmd#1\%}
\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}}\fi\global\expandafter\let\csname\cf@encoding \string#1\expandafter\endcsname\csname ?\string#1\endcsname\fi\csname\cf@encoding\string#1\expandafter\endcsname\else\noexpand#1\fi\fi}
\def\@changed@x@err#1{\errhelp{Your command will be ignored, type <return> to proceed}\errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
\def\DeclareTextCommandDefault#1{\DeclareTextCommand#1?}
\def\ProvideTextCommandDefault#1{\ProvideTextCommand#1?}
Currently we only use the \LaTeX method for accents for those that are known to be made active in some language definition file.

\input babel.def

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

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

And a few more “dummy” definitions.

A proxy file:

\input babel.def

\section{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