Babel

Localization and internationalization

Unicode
\TeX
pdf\TeX
Lua\TeX
Xe\TeX
## Contents

1 Identification and loading of required files 3  
2 locale directory 3  
3 Tools 3  
  3.1 Multiple languages 7  
  3.2 The Package File (\input{\jobname}, babel.sty) 8  
  3.3 base 9  
  3.4 key=value options and other general option 10  
  3.5 Conditional loading of shorthands 11  
  3.6 Interlude for Plain 13  
4 Multiple languages 13  
  4.1 Selecting the language 15  
  4.2 Errors 23  
  4.3 Hooks 25  
  4.4 Setting up language files 27  
  4.5 Shorthands 29  
  4.6 Language attributes 38  
  4.7 Support for saving macro definitions 40  
  4.8 Short tags 41  
  4.9 Hyphens 42  
  4.10 Multiencoding strings 43  
  4.11 Macros common to a number of languages 49  
  4.12 Making glyphs available 49  
    4.12.1 Quotation marks 49  
    4.12.2 Letters 51  
    4.12.3 Shorthands for quotation marks 52  
    4.12.4 Umlauts and tremas 52  
  4.13 Layout 54  
  4.14 Load engine specific macros 54  
  4.15 Creating and modifying languages 55  
5 Adjusting the Babel behavior 77  
  5.1 Cross referencing macros 79  
  5.2 Marks 82  
  5.3 Preventing clashes with other packages 82  
    5.3.1 ifthen 82  
    5.3.2 varioref 83  
    5.3.3 hhline 84  
  5.4 Encoding and fonts 84  
  5.5 Basic bidi support 86  
  5.6 Local Language Configuration 89  
  5.7 Language options 89  
6 The kernel of Babel (babel.def, common) 93  
7 Loading hyphenation patterns 93  
8 Font handling with fontspec 97  
9 Hooks for XeTeX and LuaTeX 100  
  9.1 XeTeX 100  
  9.2 Layout 102  
  9.3 8-bit TeX 104  
  9.4 LuaTeX 104  
  9.5 Southeast Asian scripts 110  
  9.6 CJK line breaking 112
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

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

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

```latex
⟨⟨Basic macros⟩⟩ ≡
\bbl@trace{Basic macros}
\def\bbl@add#1#2{\bbl@ifset{\bbl@stripslash#1}{\def#1{#2}}{\expandafter\def\expandafter#1\expandafter{#1#2}}}
\def\bbl@xins@{\@expandtwoargs\xins@}
\def\bbl@cargs#1#2{\expandafter#1\csname#2\endcsname}
\def\bbl@ncargs#1#2#3{\expandafter#1\expandafter#2\csname#3\endcsname}
\def\bbl@ccargs#1#2#3{\expandafter#1\csname#2\expandafter\endcsname\csname#3\endcsname}
\def\bbl@csargs#1{\expandafter\csname bbl@#1\endcsname}
\def\bbl@cl#1{\csname bbl@#1@languagename\endcsname}
```
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.

\bbl@add@list
\bbl@afterelse
\bbl@afterfi
\bbl@exp
\bbl@trim
\bbl@ifunset

18-19 \def\bbl@loop#1#2#3{\bbl@loop#1{#3}#2,\@nil,}
20 \def\bbl@loopx#1#2{\expandafter\bbl@loop\expandafter#1\expandafter{#2}}
21 \ifx\@nil#3\relax\else
22 \bbl@loopx#3\bbl@afterfi\bbl@loopx#1(#2)\% 
23 \fi
24 \def\bbl@for#1#2#3{\bbl@loopx#1{#2}{\ifx#1\@empty\else#3\fi}}

This code is based on code presented in TUGboat vol. 12, no2, June 1991 in “An expansion Power Lemma” by Sonja Maus.

---

\bbl@add@list
\bbl@afterelse
\bbl@afterfi
\bbl@exp
\bbl@trim
\bbl@ifunset

---

1This code is based on code presented in TUGboat vol. 12, no2, June 1991 in “An expansion Power Lemma” by Sonja Maus.
memory. Defined inside a group, to avoid \ifcsname being implicitly set to \relax by the \csname test.

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

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

76 \def\bbl@ifblank#1{\
77 \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}\
78 \long\def\bbl@ifblank@i#1#2\@nil#3#4\@nil{#4}\
79 \def\bbl@ifset#1#2#3{\
80 \bbl@ifunset{#1}{#3}{\bbl@exp{\@nameuse{#1}}{#3}{#2}}}\

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

81 \def\bbl@forkv#1#2{\
82 \def\bbl@kvcmd##1##2##3{#2}\
83 \bbl@kvnext#1,\@nil,}\
84 \def\bbl@kvnext#1,{\
85 \ifx\@nil#1\relax\else\
86 \bbl@ifblank{#1}{}{\bbl@forkv=\@empty=\@nil={#1}}\
87 \expandafter\bbl@kvnext\
88 \fi\
89 \def\bbl@forkv=\@empty=\@nil={#2}\
90 \def\bbl@for(cmd=\@forkv@#1=\@empty=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@nil=\@ni...
An extension to the previous macro. It takes into account the parameters, and it is string based (ie, if you replace \relax by ho, then \relax becomes \rho). No checking is done at all, because it is not a general purpose macro, and it is used by babel only when it works (an example where it does not work is in \bbl@TG@date, and also fails if there are macros with spaces, because they are retokenized). It may change! (or even merged with \bbl@replace; I'm not sure checking the replacement is really necessary or just paranoia).

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

Another hackish tool, to apply case changes inside a protected macro. 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

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

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

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

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

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

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

```
\ifpackagewith{babel}{debug}
\let\bbl@debug\@firstofone
\ifx\directlua\@undefined\else
\directlua{ Babel = Babel or {}\n Babel.debug = true }%
\input{babel-debug.tex}%
\fi
\fi
\def\bbl@error#1#2{\begingroup\def\MessageBreak\PackageError{babel}{#1}{#2}\endgroup}
\def\bbl@warning#1{\begingroup\def\MessageBreak\PackageWarning{babel}{#1}\endgroup}
\def\bbl@infowarn#1{\begingroup\def\MessageBreak\PackageNote{babel}{#1}\endgroup}
\def\bbl@info#1{\begingroup\def\MessageBreak\PackageInfo{babel}{#1}\endgroup}
```

This file also takes care of a number of compatibility issues with other packages and defines a few additional package options. Apart from all the language options below we also have a few options that influence the behavior of language definition files.

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

\ifpackagewith{babel}{silent}
{\let\bbl@info\@gobble
 \let\bbl@infowarn\@gobble
 \let\bbl@warning\@gobble}
{}
\%
\def\AfterBabelLanguage#1{%
\global\expandafter\bbl@add\csname#1.ldf-h@@k\endcsname}

If the format created a list of loaded languages (in \bbl@languages), get the name of the 0-th to show the actual language used. Also available with base, because it just shows info.
\ifx\bbl@languages\undefined\else
\begingroup
\catcode`\^^I=12
\@ifpackagewith{babel}{showlanguages}{%
\begingroup
\def\bbl@elt#1#2#3#4{\wlog{#2\^^I#1\^^I#3\^^I#4}}%
\wlog{<*languages>}%
\bbl@languages%
\wlog{</languages>}%
\endgroup}{}
\endgroup
\def\bbl@elt#1#2#3#4{%\ifnum#2=\z@
\gdef\bbl@nulllanguage{#1}%
\def\bbl@elt##1##2##3##4{}%
\fi}%
\bbl@languages%
\fi%

3.3 base
The first ‘real’ option to be processed is base, which set the hyphenation patterns then resets \ver@babel.sty so that \overleaf forgets about the first loading. After a subset of babel.def has been loaded (the old switch.def) and \AfterBabelLanguage defined, it exits.
Now the base option. With it we can define (and load, with \luatex) hyphenation patterns, even if we are not interested in the rest of babel.
3.4  key=value options and other general option

The following macros extract language modifiers, and only real package options are kept in the option list. Modifiers are saved and assigned to \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.

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

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

```latex
\DeclareOption{KeepShorthandsActive}{}
\DeclareOption{activeacute}{}
\DeclareOption{activegrave}{}
\DeclareOption{debug}{}
\DeclareOption{noconfigs}{}
\DeclareOption{showlanguages}{}
\DeclareOption{silent}{}
% \DeclareOption{mono}{}
\DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
\chardef\bbl@iniflag\z@
\DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main -> +1
\DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@} % add = 2
\DeclareOption{provide**}{\chardef\bbl@iniflag\thr@@} % add + main
% A separate option
\newif\bbl@single
\DeclareOption{selectors=off}{\bbl@singletrue}
```

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

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

The following tool is defined temporarily to store the values of options.
\def\bbl@tempa#1=#2{\bbl@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
\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 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@\bbl@exp{\\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{\in@{,provide,}{,#1,}}\ifin@\def\bbl@opt@provide{#2}\bbl@replace\bbl@opt@provide{;}{,}\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}}}{\in@{,provide,}{,#1,}}\ifin@\bbl@replace\bbl@opt@provide{;}{,}\fi
3.5 Conditional loading of shorthands
If there is no shorthands=<chars>, the original babel macros are left untouched, but if there is, these
macros are wrapped (in babel.def) to define only those given.
A bit of optimization: if there is no shorthands=, then \bbl@ifshorthand is always true, and it is
always false if shorthands is empty. Also, some code makes sense only with shorthands=....
\bbl@trace{Conditional loading of shorthands}
\def\bbl@sh@string#1{\%\ifx#1\empty\else\ifx#1\string=~\%\else\string#1\%\fi
\else\string#1\string=\%\else\string#1\fi
\bbl@trace{Conditional loading of shorthands}
\def\bbl@sh@string#1{\%\ifx#1\empty\else\ifx#1\string=~\%\else\string#1\%\fi
\else\string#1\string=\%\else\string#1\fi
The following macro tests if a shorthand is one of the allowed ones.

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

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

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

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

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

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.

\newcommand\IfBabelLayout[3]{\bbl@exp{\\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{\in@{,layout,}{,#1,}\ifin@\def\bbl@opt@layout{#2}\bbl@replace\bbl@opt@layout{ }{.}\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.

\newcommand\IfBabelLayout[1]{\@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}\ifin@\expandafter\@firstoftwo\else\expandafter\@secondoftwo\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*.

```latex
\ifx\ldf@quit\@undefined\else\endinput\fi % Same line!
\ProvidesFile{babel.def}\[\langle\langle\MakeMakeLaTeX\rangle\rangle\]
\def\bbl@version{\langle\langle\version\rangle\rangle} \def\bbl@date{\langle\langle\date\rangle\rangle}
\langle\langle\DefineCoreSwitchingMacros\rangle\rangle
\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.

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

4 Multiple languages

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

\bbl@version\bbl@date
\langle\langle\DefineCoreSwitchingMacros\rangle\rangle
\adddialect

\bbl@fixname 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 \MakeXXX case. However, a lowercase form is not imposed to improve backward compatibility (perhaps you defined a language named MYLANG, but unfortunately mixed case names cannot be trapped). Note l@ is encapsulated, so that its case does not change.
After a name has been ‘fixed’, the selectors will try to load the language. If even the fixed name is not defined, will load it on the fly, either based on its name, or if activated, its BCP47 code.

We first need a couple of macros for a simple BCP 47 lookup. It also makes sure, with \bbl@bcpcase, casing is the correct one, so that sr-latn-ba becomes fr-Latn-BA. Note #4 may contain some \@empty’s, but they are eventually removed. \bbl@bcplookup either returns the found ini or it is \relax.

\def\bbl@bcplookup{\let\bbl@bcp\relax\lowercase{\def\bbl@tempa{#1}}\ifx\@empty#2\else\bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb\bbl@bcpcase#3\@empty\@empty\@@\bbl@tempc\bbl@bcpcase#4\@empty\@empty\@@\bbl@tempd\fi}
\IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}\IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}{\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}\IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}{\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}\IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}{\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}\IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}\fi
\iffalse\empty@\empty@\empty@\relax\else\uppercasenolowercase{\def#1{}\
\let\bbl@bcp\relax\lowercase{\def\bbl@tempa{#1}}\ifx\@empty#2\else\bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb\bbl@bcpcase#3\@empty\@empty\@@\bbl@tempc\bbl@bcpcase#4\@empty\@empty\@@\bbl@tempd\fi}
\IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}\IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}{\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}\IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}{\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}\IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}{\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}\IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}\fi
\def\bbl@provide@locale{%  \ifx\babelprovide\@undefined
    \bbl@error{For a language to be defined on the fly 'base'\%
    is not enough, and the whole package must be\%
    loaded. Either delete the 'base' option or\%
    request the languages explicitly}%
    {See the manual for further details.}%
  \fi
  \let\bbl@auxname\languagename % Still necessary. TODO
  \bbl@ifunset{bbl@bcp@map@\languagename}{}% Move uplevel??
  \ifbbl@bcpallowed
  \expandafter\ifx\csname date\languagename\endcsname\relax
    \expandafter
    \bbl@bcplookup\languagename-\@empty-\@empty-\@empty\@@
    \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
      \edef\languagename{\bbl@bcp@prefix\bbl@bcp}\
      \edef\localename{\bbl@bcp@prefix\bbl@bcp}\
      \expandafter\ifx\csname date\languagename\endcsname\relax
        \let\bbl@initoload\bbl@bcp
        \bbl@exp{\
            \babelprovide[\bbl@autoload@bcpoptions]{\languagename}}\
        \let\bbl@initoload\relax
      \fi
    \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}\
  \fi
  \fi
  \fi
  \expandafter\ifx\csname date\languagename\endcsname\relax
    \IfFileExists{babel-\languagename.tex}\
    {\bbl@exp{\
        \babelprovide[\bbl@autoload@options]{\languagename}}}\
    {}\
  \fi}

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

\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.

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

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

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

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

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

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

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

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

\begin{verbatim}
\def\bbl@set@language{\expandafter\bbl@set@language\expandafter{\languagename}}
\end{verbatim}

The reason for the somewhat weird arrangement of arguments to the helper function is the fact it is called in the following way. This means that before \texttt{\bbl@pop@lang} is executed \TeX{} first \texttt{expands} the stack, stored in \texttt{\bbl@language@stack}. The result of that is that the argument string of \texttt{\bbl@pop@lang} contains one or more language names, each followed by a '+'-sign (zero language names won't occur as this macro will only be called after something has been pushed on the stack).

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

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

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

\begin{verbatim}
\chardef\localeid\z@
\def\bbl@id@last{0} % No real need for a new counter
\def\bbl@id@assign{%
  \bbl@ifunset{bbl@id@@\languagename}%
  {\count@\bbl@id@last\relax
\end{verbatim}

16
The unprotected part of `\selectlanguage`.

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

We also write a command to change the current language in the auxiliary files. `\bbl@savelastskip` is used to deal with skips before the write whatsis (as suggested by U Fischer).

Adapted from `hyperref`, but it might fail, so I'll consider it a temporary hack, while I study other options (the ideal, but very likely unfeasible except perhaps in `luatex`, is to avoid the `\write` altogether when not needed).
First, check if the user asks for a known language. If so, update the value of \language and call \originalTeX to bring TeX in a certain pre-defined state. The name of the language is stored in the control sequence \languagename. Then we have to redefine \originalTeX to compensate for the things that have been activated. To save memory space for the macro definition of \originalTeX, we construct the control sequence name for the \noextras (lang) command at definition time by expanding the \csname primitive. Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of \selectlanguage, and calling these macros.

The switching of the values of \lefthyphenmin and \righthyphenmin is somewhat different. First we save their current values, then we check if \langle lang\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@switchempty=\@empty
\def\bbl@switch{% from select@, foreign@
% make sure there is info for the language if so requested
\bbl@ensureinfo[#1]%
% restore \originalTeX
\expandafter\def\expandafter\originalTeX\expandafter{%
\csname noextras\endcsname
% set the locale id
\bbl@id@assign
%
switch captions, date
\bbl@bsphack
\ifcase\bbl@select@type
\csname captions\select@opt\endcsname\relax
\csname date\select@opt\endcsname\relax
\else
\bbl@xin\{\captionsopts\},\bbl@select@opts%\ifin@
\csname captions\select@opt\endcsname\relax
\fi
\bbl@xin\{\dateopts\},\bbl@select@opts%\ifin@ % if \foreign... within \lang\date
\csname date\select@opt\endcsname\relax
\fi
\fi
\bbl@esphack
%
switch extras
\csname bbl@preextras\select@opt\endcsname
\bbl@usehooks{beforeextras}{}
\csname extras\select@opt\endcsname\relax
\bbl@usehooks{afterextras}{}
% > babel-ensure
% > babel-sh-<short>
% > babel-bidi
% > babel-fontspec
\let\bbl@savedextras\@empty
% hyphenation - case mapping
\ifcase\bbl@opt@hyphenmap\or
\def\BabelLower##1##2\lccode##1=##2\relax%
\ifnum\bbl@hymapsel>4\else
\csname\languagename @bbl@hyphenmap\endcsname
\fi
\chardef\bbl@opt@hyphenmap\z@
\else
\ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
\csname\languagename @bbl@hyphenmap\endcsname
\fi
\let\bbl@hymapsel\@cclv
% hyphenation - select rules
\ifnum\languagename =\l@unhyphenated
\edef\bbl@tempa{u}%
\else
\edef\bbl@tempa{\bbl@cl{lnbrk}}%
\fi
% linebreaking - handle u, e, k (v in the future)
\ifin@\bbl@xin\{u\},\bbl@tempa%\fi % elongated forms
\ifin@\bbl@xin\{e\},\bbl@tempa%\fi % only kashida
\ifin@\bbl@xin\{p\},\bbl@tempa%\fi % padding (eg, Tibetan)
\ifin@\bbl@xin\{v\},\bbl@tempa%\fi % variable font
\ifin@
% unhyphenated/kashida/elongated/padding = allow stretching
\language\l@unhyphenated
\bbl@savevariable\emergencystretch
\emergencystretch\maxdimen
\bbl@savevariable\hbadness\z@
\else
% other = select patterns
\bbl@patterns\select@opt%\fi

otherlanguage (env.) The otherlanguage environment can be used as an alternative to using the \selectlanguage declarative command. When you are typesetting a document which mixes left-to-right and right-to-left typesetting you have to use this environment in order to let things work as you expect them to.

The \ignorespaces command is necessary to hide the environment when it is entered in horizontal mode.

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

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

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 \extrans{lang} command doesn’t make any \global changes. The coding is very similar to part of \selectlanguage.

\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.

This macro does the work for `\foreignlanguage` and the other `\language*` environment. First we need to store the name of the language and check that it is a known language. Then it just calls `bbl@switch`.

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

This macro selects the hyphenation patterns by changing the \language register. If special hyphenation patterns are available specifically for the current font encoding, use them instead of the default. It also sets hyphenation exceptions, but only once, because they are global (here language \lccode's has been set, too). `\bbl@hyphenation@` is set to relax until the very first `\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=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
  \csname l@#1\endcsname
  \edef\bbl@tempa{#1}%
  \else
  \csname l@#1:\f@encoding\endcsname
  \edef\bbl@tempa{#1:\f@encoding}%
  \fi
  \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
% > luatex
  \@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
\begingroup
\bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
\ifin@\else
  \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
  \hyphenation{\bbl@hyphenation@
    \ifundefined{bbl@hyphenation@#1}{}{\space\csname bbl@hyphenation@#1\endcsname}}%
  \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
\fi
\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, \lccode's and font encodings are not set at all, so in most cases you should use otherlanguage*.

\begin{verbatim}
\def\hyphenrules#1{%
  \edef\bbl@tempf{#1}%
  \bbl@fixname\bbl@tempf
  \bbl@iflanguage\bbl@tempf{%
    \expandafter\expandafter\expandafter\set@hyphenmins
    \csname\bbl@tempf hyphenmins\endcsname\relax
    \if\languageshorthands\@undefined\else
      \languageshorthands{none}%
    \fi
    \expandafter\expandafter\expandafter\set@hyphenmins
    \csname\bbl@tempf hyphenmins\endcsname\relax
    \if\languageshorthands\@undefined\else
      \languageshorthands{none}%
    \fi
  \fi
  \expandafter\expandafter\expandafter\set@hyphenmins
  \csname\bbl@tempf hyphenmins\endcsname\relax
  \set@hyphenmins{tw@\thr@@}\relax
  \else
    \expandafter\set@hyphenmins\expandafter\expandafter\expandafter\set@hyphenmins
  \fi%
\end{verbatim}

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

\begin{verbatim}
\def\providehyphenmins#1#2{%
  \edef\bbl@tempf{#1}%
  \bbl@fixname\bbl@tempf
  \expandafter\@namedef{\bbl@tempf hyphenmins}{#2}%
  \fi}
\end{verbatim}

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

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

\[\text{\providecommand}\ProvidesLanguage{\@empty}\]

Becausethispartofthecodecanbeincludedinaformat,wemakesurethatthemacrowhichinitializesthesavemechanism,\babel@beginsave, is not considered to be undefined.

\[\text{\providecommand}\setlocale{\@empty}\]

A few macro names are reserved for future releases of babel, which will use the concept of 'locale':

\[\text{\providecommand}\setlocale{\@empty}\]

\subsection*{4.2 Errors}

The babel package will signal an error when a documents tries to select a language that hasn’t been defined earlier. When a user selects a language for which no hyphenation patterns were loaded into the format he will be given a warning about that fact. We revert to the patterns for \language=0 in that case. In most formats that will be (US)english, but it might also be empty.
\edef\bbl@tempa{#1}\
\bbl@sreplace\bbl@tempa{name}{}\
\bbl@warning{\
\@backslashchar#1 not set for '\languagename'. Please,\
define it after the language has been loaded\
(typically in the preamble) with: \\n\string\setlocalecaption{\languagename}\{\bbl@tempa\}{}\
Feel free to contribute on github.com/latex3/babel.\
Reported}}
\def\bbl@tentative{\protect\bbl@tentative@i}
\def\bbl@tentative@i#1{\
\bbl@warning{\
Some functions for '#1' are tentative.\
They might not work as expected and their behavior\
could change in the future.\
Reported}}
\def\@nolanerr#1{\
\bbl@error{You haven't defined the language '#1' yet.\
Perhaps you misspelled it or your installation\
is not complete}\
(Your command will be ignored, type <return> to proceed)}
\def\@nopatterns#1{\
\bbl@warning{No hyphenation patterns were preloaded for\'
the language '#1' into the format.\
Please, configure your TeX system to add them and\nrebuild the format. Now I will use the patterns\npreloaded for \bbl@nulllanguage\space instead}}
\let\bbl@usehooks\@gobbletwo
\ifx\bbl@onlyswitch\@empty\endinput\fi
\ifx\directlua\@undefined\else
\ifx\bbl@luapatterns\@undefined
\input luababel.def
\fi\fi
\bbl@trace{Compatibility with language.def}
\ifx\bbl@languages\@undefined
\ifx\directlua\@undefined
\openin1 = language.def % TODO. Remove hardcoded number
\ifeof1
\closein1
\message{I couldn’t find the file language.def}
\else
\closein1
\begingroup
\def\addlanguage#1#2#3#4#5{\
\expandafter\ifx\csname lang@#1\endcsname\relax\else
\global\expandafter\let\csname l@#1\expandafter\endcsname
\csname lang@#1\endcsname
\fi}
\def\uselanguage#1{}\
\input language.def
\endgroup
\fi
\fi
\fi
\addto It takes two arguments, a ⟨control sequence⟩ and TeX-code to be added to the ⟨control sequence⟩.
If the ⟨control sequence⟩ has not been defined before it is defined now. The control sequence could also expand to \relax, in which case a circular definition results. The net result is a stack overflow. Note there is an inconsistency, because the assignment in the last branch is global.

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

The macro \initiateactivechar 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{\begin{group}\lccode`~=`#2\relax\lowercase{\end{group}#1~}}

\bbl@redefine To redefine a command, we save the old meaning of the macro. Then we redefine it to call the original macro with the ‘sanitized’ argument. The reason why we do it this way is that we don’t want to redefine the \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 \orgmacro.

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

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

\def\bbl@redefinerobust#1{\edef\bbl@tempa{\bbl@stripslash#1}\bbl@ifunset{\bbl@tempa \space}{\expandafter\let\csname org@\bbl@tempa\endcsname#1\expandafter\def\csname \bbl@tempa \space\endcsname}}} \bbl@usehooks is the commands used by babel to execute hooks defined for an event.

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.

\bbl@trace\{Hooks\}
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).

The babelensure command just parses the optional argument and creates a new macro named \bbl@ensure\langle\textit{language}\rangle. 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@ensure\langle\textit{language}\rangle contains \bbl@ensure\langle\textit{include}\rangle\langle\textit{exclude}\rangle\langle\textit{fontenc}\rangle, which in turn loops over the macros names in \bbl@captionslist, excluding (with the help of \in@) those in the exclude list. If the \textit{fontenc} is given (and not \relax), the \textit{fontenc} is also added. Then we loop over the include list, but if the macro already contains \foreignlanguage\langle\textit{language}\rangle, 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`.

```
1106 \bbl@trace{Macros for setting language files up}
1107 \def\bbl@ldfinit{%
1108 \let\bbl@screset\empty
1109 \let\BabelStrings\bbl@opt@string
1110 \let\BabelOptions\@empty
1111 \let\BabelLanguages\relax
1112 \ifx\originalTeX\@undefined
1113 \let\originalTeX\@empty
1114 \else
1115 \originalTeX
1116 \fi}
1117 \def\LdfInit#1#2{%
1118 \chardef\atcatcode=\catcode`@\relax
1119 \catcode`@=11 \relax
1120 \chardef\eqcatcode=\catcode`\=\relax
1121 \catcode`\==12 \relax
1122 \expandafter\if\expandafter\@backslashchar
1123 \expandafter\@car\string#2\@nil
1124 \ifx#2\@undefined\else
1125 \ldf@quit{#1}%
1126 \fi
1127 \else
1128 \expandafter\ifx\csname#2\endcsname\relax\else
1129 \ldf@quit{#1}%
1130 \fi
1131 \fi
1132 \bbl@ldfinit}
1133 \ldf@quit
1134 \def\ldf@quit#1{%
1135 \expandafter\main@language\expandafter{#1}%
1136 \catcode`\@=\atcatcode \let\atcatcode\relax
1137 \catcode`\==\eqcatcode \let\eqcatcode\relax
1138 \endinput
```

This macro interrupts the processing of a language definition file.

```
1139 \def\ldf@quit#1{%
1140 \expandafter\main@language\expandafter{#1}%
1141 \catcode`\@=\atcatcode \let\atcatcode\relax
1142 \catcode`\==\eqcatcode \let\eqcatcode\relax
1143 \endinput
```

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.

```
1144 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1145 \bbl@afterlang
1146 \let\bbl@afterlang\relax
1147 \let\BabelModifiers\relax
1148 \let\bbl@screset\relax%
1149 \def\ldf@finish#1{%
1150 \loadlocalcfg{#1}%
1151 \bbl@afterldf{#1}%
1152 \expandafter\main@language\expandafter{#1}%
1153 \catcode`\@=\atcatcode \let\atcatcode\relax
1154 \catcode`\==\eqcatcode \let\eqcatcode\relax
```

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

```
1155 \onlypreamble\LdfInit
1156 \onlypreamble\ldf@quit
1157 \onlypreamble\ldf@finish
```

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

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

We also have to make sure that some code gets executed at the beginning of the document, either when the 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}
def\bbl@beforestart{% 
def\@nolanerr##1{% 
\bbl@warning{Undefined language '##1' in aux.\Reported}}%
def\usehooks{beforestart}{}% 
global\let\bbl@beforestart=\relax
\AtBeginDocument{% 
{\@nameuse{bbl@beforestart}}% Group! 
@if\files\providecommand\babel@aux[2]{}% 
immediate\write\@mainaux{\providecommand\babel@aux[2]{}}% 
immediate\write\@mainaux{\@nameuse{bbl@beforestart}}% 
\fi
\expandafter\selectlanguage\expandafter{\bbl@main@language}%
\fi
\fi\end{verbatim}

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

\begin{verbatim}
def\select@language@x#1{% 
\ifcase\select@type
\bbl@ifsamestring\languagename{#1}{}{\select@language{#1}}%
\else
\select@language{#1}%
\fi
\expandafter\selectlanguage\expandafter{\bbl@main@language}%
\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 \fss@catcodes, added in 3.10.

\begin{verbatim}
def\bbl@add@special#1{% 
def\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
def\if\files\ifcase\bbl@engine
\AtBeginDocument{\pagedir\bodydir} % TODO - a better place
\fi
\fi
\ifcase\bbl@single
\renewcommand\selectlanguage[1]{}%
\renewcommand\foreignlanguage[2]{#2}%
global\let\babel@aux[2]{}% Also as flag
\fi
\fi
\begingroup
\edef\bbl@add@special#1{% 
\bbl@add\dospecials{% 
\bbl@ifsamestring\languagename{#1}{\select@language{#1}}%
\else
\select@language{#1}%
\fi
\end{verbatim}

4.5 Shorthands

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

\bbl@active@char

A language definition file can call this macro to make a character active. This macro takes one argument, the character that is to be made active. When the character was already active this macro does nothing. Otherwise, this macro defines the control sequence \normal@char{char} to expand to the character in its ‘normal state’ and it defines the active character to expand to \normal@char{char} by default (\normal@char{char} being the character to be made active). Later its definition can be changed to expand to active@char{char} by calling \bbl@activate{char}.

For example, to make the double quote character active one could have \bbl@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).

\bbl@active@def

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.

\bbl@active@char calls \bbl@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.

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

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

To prevent problems with the loading of other packages after babel we reset the catcode of the character to the original one at the end of the package and of each language file (except with \KeepShorthandsActive). It is re-active again at \begin{document}. We also need to make sure that the shorthands are active during the processing of the .aux file. Otherwise some citations may give unexpected results in the printout when a shorthand was used in the optional argument of \bibitem for example. Then we make it active (not strictly necessary, but done for backward compatibility).

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

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

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

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

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

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

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

Initiating a shorthand makes active the char. That is not strictly necessary but it is still done for backward compatibility. So we need to restore the original catcode at the end of package and the end of the ldf.
This command helps the shorthand supporting macros to select how to proceed. Note that this macro needs to be expandable as do all the shorthand macros in order for them to work in expansion-only environments such as the argument of \hyphenation. This macro expects the name of a group of shorthands in its first argument and a shorthand character in its second argument. It will expand to either \bbl@firstcs or \bbl@scndcs. Hence two more arguments need to follow it.

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

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

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

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

\chardef\bbl@activated\z@
\def\bbl@activate#1{\chardef\bbl@activated\@ne\bbl@withactive\let\expandafter\csname bbl@active@\string#1\endcsname}
\def\bbl@deactivate#1{\chardef\bbl@activated\tw@\bbl@withactive\let\expandafter\csname bbl@normal@\string#1\endcsname}
\bbl@firstcs\bbl@scndcs
These macros are used only as a trick when declaring shorthands.

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

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

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

\def\babel@texpdf#1#2#3#4{%
  \ifx\textorpdfstring\@undefined\textormath{#1}{#3}%
  \else\texorpdfstring{\textormath{#1}{#3}}{#2}%
  \fi}
\def\declare@shorthand#1#2#3#4{%
  \@decl@short{#1}{#2}{#3}{#4}
}
\def\@decl@short#1#2#3#4{%
  \def\bbl@tempa{#3}
  \ifx\bbl@tempa\@empty
    \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
    \bbl@ifunset{#1@sh@\string#2@}{}%
  {\def\bbl@tempa{#4}
    \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
    \else
      \bbl@info{Redefining #1 shorthand \string#2\%
        in language \CurrentOption}%
    \fi}
  \@namedef{#1@sh@\string#2@}{#4}%
  \else
  \fi}
\def\@decl@short#1#2#3#4{%
  \@decl@short{#1}{#2}{#3}{#4}
}
\textormath  Some of the shorthands that will be declared by the language definition files have to be usable in both text and math mode. To achieve this the helper macro \textormath is provided.

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

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

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

\useshorthands
\begin{verbatim}
\def\useshorthands{%
  \@ifstar\bbl@usesh@s{bl@usesh@x}{}
\end{verbatim}

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

\defineshorthand
\begin{verbatim}
\def\defineshorthand{\@ifstar\bbl@definesh@s{\bbl@definesh@x}{}%}
\end{verbatim}

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

\languageshorthands A user level command to change the language from which shorthands are used. Unfortunately, \texttt{babel} currently does not keep track of defined groups, and therefore there is no way to catch a possible change in casing to fix it in the same way languages names are fixed. [TODO].

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

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

\shorthandoff The macro \texttt{\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 \texttt{\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 \texttt{\active@char"} should exist. Switching off and on is easy – we just set the category code to ‘other’ (12) and \texttt{\active}. With the starred version, the original catcode and the original definition, saved in \texttt{\@initiate@active@char}, are restored.
(I can't switch 'string2' on or off--not a shorthand)%
(This character is not a shorthand. Maybe you made a typing mistake? I will ignore your instruction.))%
{\ifcase#1% off, on, off*
  \catcode\string212\relax
\or
  \catcode\string2\active
  \bbl@ifunset{bbl@shdef@string2}{}
  \bbl@withactive{\expandafter\let\expandafter}\string2\relax%
  \csname bbl@csarg\let{shdef\string2}\relax%
  \ifcase\bbl@activated\or
    \bbl@activate{#2}%
  \else
    \bbl@deactivate{#2}%
  \fi
}\bl@afterfi\bbl@switch@sh#1%
\fi
Notethevalueisthatattheexpansiontime; e.g., in the preexample shorthands are usually deactivated.
\def\bblshorthand{\active@prefix\bblshorthand\bbl@putsh}
\def\bbl@putsh#1{\bbl@ifunset{bbl@active@string1}{\bbl@putsh@i#1\@empty\@nnil}{\csname bbl@active@string1\endcsname}}
\def\bbl@putsh@i#1#2\@nnil{\csname\language@group@sh@string1@%\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@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
  \let\bbl@s@activate\bbl@activate
  \def\bbl@activate#1{\bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
  \let\bbl@s@deactivate\bbl@deactivate
  \def\bbl@deactivate#1{\bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
\fi
You may want to test if a character is a shorthand. Note it does not test whether the shorthand is on or off.
\newcommand\ifbblshorthand[3]{{\bbl@ifunset{bbl@active@string1}{#3}{#2}}
One of the internal macros that are involved in substituting \prime for each right quote in math mode is \bbl@prim@s. This checks if the next character is a right quote. When the right quote is active, the definition of this macro needs to be adapted to look also for an active right quote; the hat could be active, too.
Usually the ~ is active and expanded to \penalty\@M \penalty 4444. When it is written to the .aux file it is written expanded. To prevent that and to be able to use the character ~ as a start character for a shorthand, it is redefined here as a one character shorthand on system level. The system declaration is in most cases redundant (when ~ is still a non-break space), and in some cases is inconvenient (if ~ has been redefined); however, for backward compatibility it is maintained (some existing documents may rely on the babel value).

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

\OT1dqpos
\T1dqpos

The position of the double quote character is different for the OT1 and T1 encodings. It will later be selected using the \f@encoding macro. Therefore we define two macros here to store the position of the character in these encodings.
\expandafter\def\csname OT1dqpos\endcsname{127}
\expandafter\def\csname T1dqpos\endcsname{4}

When the macro \f@encoding is undefined (as it is in plain \TeX) we define it here to expand to OT1
\ifx\f@encoding\@undefined
  \def\f@encoding{OT1}
\fi

4.6 Language attributes

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

The macro \languageattribute checks whether its arguments are valid and then activates the selected language attribute. First check whether the language is known, and then process each attribute in the list.
\bbl@trace{Language attributes}
\newcommand\languageattribute[2]{%}
  \def\bbl@tempc{#1}%
  \bbl@fixname\bbl@tempc
  \bbl@iflanguage\bbl@tempc{%
    \bbl@vforeach{#2}{%
      \bf\bbl@tempc-##1,}{,\bbl@known@attribs,}%
    \bbl@known@attribs%
  \bbl@fixname\bbl@tempc%
  }%
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.

\bbl@warning{You have more than once selected the attribute '##1' for language #1. Reported}\else
When we end up here the attribute is not selected before. So, we add it to the list of selected attributes and execute the associated \TeX-code.

\bbl@exp{\bbl@add@list\bbl@known@attribs{\bbl@tempc-##1}}\edef\bbl@tempa{\bbl@tempc-##1}\expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes{%\csname\bbl@tempc @attr@##1\endcsname}{\@attrerr{\bbl@tempc}{##1}}\fi}}\fi
\@onlypreamble\languageattribute

The error text to be issued when an unknown attribute is selected.
\newcommand*{\@attrerr}[2]{\bbl@error{The attribute #2 is unknown for language #1.}{}{Your command will be ignored, type <return> to proceed}}\bbl@declare@ttribute

This command adds the new language/attribute combination to the list of known attributes. Then it defines a control sequence to be executed when the attribute is used in a document. The result of this should be that the macro \extras... for the current language is extended, otherwise the attribute will not work as its code is removed from memory at \begin{document}.
\def\bbl@declare@ttribute#1#2#3{\bbl@xin{,#2,}{,\BabelModifiers,}\ifin@\AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}\fi\bbl@add@list\bbl@attributes{#1-#2}\expandafter\def\csname#1@attr@#2\endcsname{#3}}\bbl@ifattributeset

This internal macro has 4 arguments. It can be used to interpret \TeX-code based on whether a certain attribute was set. This command should appear inside the argument to \AtBeginDocument because the attributes are set in the document preamble, after \bbl is loaded. The first argument is the language, the second argument the attribute being checked, and the third and fourth arguments are the true and false clauses.
\def\bbl@ifattributeset#1#2#3#4{%\ifx\bbl@known@attributes\undefined\in@false\else\bbl@xin{,#1-#2,}{,\bbl@known@attributes,}\ifin@\bbl@afterelse#3\else\bbl@afterfi#4\fi\fi}}\bbl@ifknown@ttrib

An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the \TeX-code to be executed when the attribute is known and the \TeX-code to be executed otherwise. We first assume the attribute is unknown. Then we loop over the list of known attributes, trying to find a match.
\def\bbl@ifknown@ttrib#1#2{%\let\bbl@tempa@@\secondoftwo\bbl@loop\bbl@loop@secondoftwo{\bbl@known@attributes}{#1}{\bbl@tempa\temp}{\expandafter\in@\expandafter{\expandafter,\bbl@tempb}}{\expandafter,\bbl@tempa\tempb\expandafter}{\expandafter,\bbl@tempa}{\expandafter,\bbl@tempb}\expandafter\ifin@}}}
\let\bbl@tempa\@firstoftwo
\else
\fi}%
\bbl@tempa}
\bbl@clear@ttribs
This macro removes all the attribute code from \LaTeX's memory at \begin{document} time (if any is present).
\def\bbl@clear@ttribs{%
\ifx\bbl@attributes\@undefined\else
\bbl@loopx\bbl@tempa{\bbl@attributes}{%$
\expandafter\bbl@clear@ttrib\bbl@tempa.}%$
\let\bbl@attributes\@undefined
\fi}
\def\bbl@clear@ttrib#1-#2.{%$
\expandafter\let\csname#1@attr@#2\endcsname\@undefined}$%
\AtBeginDocument{\bbl@clear@ttribs}

\section*{4.7 Support for saving macro definitions}
To save the meaning of control sequences using \texttt{\babel@save}, we use temporary control sequences. To save hash table entries for these control sequences, we don't use the name of the control sequence to be saved to construct the temporary name. Instead we simply use the value of a counter, which is reset to zero each time we begin to save new values. This works well because we release the saved meanings before we begin to save a new set of control sequence meanings (see \texttt{\selectlanguage} and \texttt{\originalTeX}). Note undefined macros are not undefined any more when saved – they are \texttt{\relax}'ed.

\begin{center}
\texttt{\babel@savecnt: The initialization of a new save cycle: reset the counter to zero.}
\end{center}
\begin{center}
\texttt{\babel@beginsave: Before it's forgotten, allocate the counter and initialize all.}
\end{center}
\begin{center}
\texttt{\babel@save: The macro \texttt{\babel@save}⟨csname⟩ saves the current meaning of the control sequence ⟨csname⟩ to \texttt{\originalTeX}. To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to \texttt{\originalTeX} and the counter is incremented. The macro \texttt{\babel@savevariable}⟨variable⟩ saves the value of the variable. ⟨variable⟩ can be anything allowed after the \texttt{\the} primitive. To avoid messing saved definitions up, they are saved only the very first time.}
\end{center}
\begin{center}
\texttt{\babel@savevariable: Some languages need to have \texttt{\frenchspacing} in effect. Others don't want that. The command \texttt{\bbl@frenchspacing} switches it on when it isn't already in effect and \texttt{\bbl@nonfrenchspacing} switches it off if necessary. A more refined way to switch the catcodes is done with ini files. Here an}
\end{center}

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

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

\def\bbl@trace{Short tags}
\def\bbl@tags#1{\edef\bbl@tempa{\zap@space#1 \@empty}\def\bbl@tempb##1=##2\@@{\edef\bbl@tempc{\noexpand\newcommand\expandafter\noexpand\csname ##1\endcsname{\noexpand\protect\expandafter\noexpand\csname otherlanguage*\endcsname{##2}}\noexpand\newcommand\expandafter\noexpand\csname text##1\endcsname{\foreignlanguage{##2}}}}\bbl@tempc\bbl@for\bbl@tempa\bbl@tempa{\expandafter\bbl@tempb\bbl@tempa\@@}}
4.9 Hyphens

\babelhyphenation

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

\bbatrace{Hyphens}
\@onlypreamble\babelhyphenation
\AtEndOfPackage{%
\newcommand\babelhyphenation[2][\@empty]{%
  \ifx\bbl@hyphenation@\relax
  \let\bbl@hyphenation@\@empty
  \fi
  \ifx\bbl@hyphlist\@empty\else
    \bbl@warning{%
      You must not intermingle \string\selectlanguage\space and\%
      \string\babelhyphenation\space or some exceptions will not\%
      be taken into account. Reported}%
  \fi
  \ifx\@empty#1%
    \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
  \else
    \bbl@vforeach{#1}{%
      \def\bbl@tempa{##1}%
      \bbl@fixname\bbl@tempa
      \bbl@iflanguage\bbl@tempa{%
        \protected@edef{hyphenation@\bbl@tempa}{%
          \ifunset{bbl@hyphenation@\bbl@tempa}{}%
          \csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
        #2})}%
  \fi}}
\bbl@allowhyphens

This macro makes hyphenation possible. Basically its definition is nothing more than \nobreak \hspace 0pt plus 0pt\textsuperscript{3}.

\def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
\def\bbl@t@one{T1}
\def\allowhyphens{\ifx\cf@encoding\bbl@t@one\else\bbl@allowhyphens\fi}

\babelhyphen

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

\newcommand\babelnullhyphen{\char\hyphenchar\font}
\def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
\def\bbl@hyphen{%
  \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
\def\bbl@hyphen@i#1#2{%
  \bbl@ifunset{bbl@hy@#1#2\@empty}{%
  \csname bbl@hyphenation@\bbl@tempa\endcsname{\discretionary{#2}}{{#2}}}%
  \csname bbl@hy@#1#2\@empty@endcsname}%

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

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

\def\bbl@usehyphen#1{%
  \leavevmode
  \ifdim\lastskip\z@\mbox{#1}\else\nobreak\fi
  \nobreak\hspace\z@skip}

\textsuperscript{3}\TeX begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.
The following macro inserts the hyphen char:

```tex
\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.

```tex
\def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}} 
\def\bbl@hy@@soft{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}} 
\def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar} 
\def\bbl@hy@@hard{\bbl@@usehyphen\bbl@hyphenchar} 
\def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}} 
\def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}} 
\def\bbl@hy@repeat{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}} 
\def\bbl@hy@@repeat{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}} 
\def\bbl@hy@empty{\hskip\z@skip} 
\def\bbl@hy@@empty{\discretionary{}{}{}}
```

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

```tex
\def\bbl@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bbl@allowhyphens}
```

### 4.10 Multiencoding strings

The aim following commands is to provide a common interface for strings in several encodings. They also contains several hooks which can be used by \latex 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.

```tex
\def\trace{Multiencoding strings} 
\def\trace@global#1{\global\let#1#1} 
\def\trace@globalwith#1#2{\trace@global{#1}{#2}}
```

The second one. We need to patch \ucllist, 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 \ucllist 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 \ucl. The parser is restarted inside \langle lang \rangle \bbl@ucl 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:

```tex
\let\bbl@tolower\@empty\bbl@toupper\@empty
```

and starts over (and similarly when lowercasing).

```tex
\def\patchucldetect{\@empty} 
\def\patchucldetect{\@empty} 
\def\patchucldetect{\@empty} 
\def\patchucldetect{\@empty} 
\def\patchucldetect{\@empty} 
\def\patchucldetect{\@empty} 
\def\patchucldetect{\@empty} 
\def\patchucldetect{\@empty} 
\def\patchucldetect{\@empty} 
```

43
The following package options control the behavior of \SetString.

```latex
\let\bbl@opt@strings@\@nnil \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.

```latex
\@onlypreamble\StartBabelCommands \def\StartBabelCommands{% \begingroup \@tempcnta="7F \def\bbl@tempa{% \ifnum\@tempcnta="FF\else \catcode\@tempcnta=11 \advance\@tempcnta\@ne \expandafter\bbl@tempa \fi} \bbl@tempa ⟨⟨Macros local to BabelCommands⟩⟩ \def\bbl@provstring##1##2{% \providecommand##1{##2} \bbl@toglobal##1} \global\let\bbl@scafter\@empty \let\StartBabelCommands\bbl@startcmds \if\BabelLanguages\relax \let\BabelLanguages\CurrentOption \fi \begingroup \let\bbl@screset\@nnil % local flag - disable 1st stopcommands \def\bbl@startcmds{\if\bbl@screset\@nnil\else \bbl@usehooks{stopcommands}{} \fi} \bbl@startcmds \let\bbl@startcommands\StartBabelCommands \parsetheencodinginfotogetthelabel,input,andfontparts.
```
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 (ie, fallback values). With labelled blocks and strings=encoded, define the strings, but with another value, define strings only if the current label or font encoding is the value of strings; otherwise (ie, no strings or a block whose label is not in strings=) do nothing.

We presume the current block is not loaded, and therefore set (above) a couple of default values to gobble the arguments. Then, these macros are redefined if necessary according to several parameters.

\newcommand\bbl@startcmds@ii[1][\empty]{% 
\let\SetString\@gobbletwo 
\let\bbl@stringdef\@gobbletwo 
\let\AfterBabelCommands\@gobble 
\ifx\@empty#1% 
\def\bbl@sc@label{generic}% 
\def\bbl@encstring##1##2{% 
\ProvideTextCommandDefault##1{##2}% 
\bbl@toglobal##1% 
\expandafter\bbl@toglobal\csname\string?\string##1\endcsname}% 
\let\bbl@sctest\in@true 
\else 
\let\bbl@sc@charset\space % <- zapped below 
\let\bbl@sc@fontenc\space % <- " " 
\def\bbl@tempa##1=##2@nil{% 
\bbl@csarg\edef{sc@##1@empty}{##2 }% 
\bbl@vforeach{label=#1}{\bbl@tempa##1@nil}% 
\def\bbl@tempa##1##2{% space -> comma 
  #1% 
  \ifx\@empty##2% 
  \else\fi\bbl@afterfi\bbl@tempa##2% 
}\edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc@empty}% 
\edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label@empty}% 
\edef\bbl@tempa##1=##2@nil{% 
\bbl@foreach\bbl@sc@fontenc{\bbl@ifunset{T@####1}{}{\ProvideTextCommand##1{####1}{##2}\
\bbl@toglobal##1% 
\expandafter\bbl@toglobal\csname####1\string##1\endcsname}}}% 
\def\bbl@sctest{% 
\bbl@xing{,\bbl@opt@strings,\bbl@sc@label,\bbl@sc@fontenc,}% 
\ifx\bbl@opt@strings\@nil % ie, no strings key -> defaults 
\else\fi\bbl@opt@strings\relax % ie, strings=encoded 
\let\AfterBabelCommands\bbl@aftercmds 
\let\SetString\bbl@setstring 
\let\bbl@stringdef\bbl@encstring 
\else % ie, strings=value 
\bbl@sctest 
\fi 
\ifx\bbl@opt@strings\@nil % ie, no strings key -> defaults 
\else\fi\bbl@opt@strings\relax % ie, strings=encoded 
\let\AfterBabelCommands\bbl@aftercmds 
\let\SetString\bbl@setstring 
\let\bbl@stringdef\bbl@provstring 
\if\fi\fi 
\bbl@scswitch 
\ifx\bbl@opt@G@empty 
\def\SetString##1##2{% 
\bbl@error{Missing group for string \string##1}\
{You must assign strings to some category, typically\% 
  captions or extras, but you set none})% 
\fi 
\ifx\@empty% 
\bbl@usehooks{defaultcommands}{}%
There are two versions of \bbl@scswitch. The first version is used when ldfls 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 ldfls), and the second one skips undefined languages (after babel has been loaded).

Now we define commands to be used inside \StartBabelCommands.

**Strings**

The following macro is the actual definition of \SetString when it is “active”

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

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.
\def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
\def\bbl@encoded{\relax}
\def\bbl@encoded@uclc#1{\@inmathwarn#1\expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax\expandafter\ifx\csname ?\string#1\endcsname\relax\TextSymbolUnavailable#1\else\csname ?\string#1\endcsname\fi\else\csname\cf@encoding\string#1\endcsname\fi}
\def\bbl@scset#1#2{\def#1{#2}}
\let\bbl@encoded\relax
\def\bbl@encoded@uclc#1{\@inmathwarn#1\expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax\expandafter\ifx\csname ?\string#1\endcsname\relax\TextSymbolUnavailable#1\else\csname ?\string#1\endcsname\fi\else\csname\cf@encoding\string#1\endcsname\fi}
\def\bbl@scset#1#2{\def#1{#2}}
\fi
\else
\let\bbl@encoded\relax
\def\bbl@encoded@uclc#1{\@inmathwarn#1\expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax\expandafter\ifx\csname ?\string#1\endcsname\relax\TextSymbolUnavailable#1\else\csname ?\string#1\endcsname\fi\else\csname\cf@encoding\string#1\endcsname\fi}
\def\bbl@scset#1#2{\def#1{#2}}
\fi

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.

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.

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.

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

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

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

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.

\def\SetCase[3]{\bbl@forlang\bbl@tempa{\bbl@carg\bbl@encstring{\bbl@tempa @bbl@uclc}{\bbl@tempa##1}{\bbl@tempa##2}{\bbl@tempa##3}}}%

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

There are 3 helper macros which do most of the work for you.
The following package options control the behavior of hyphenation mapping.

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

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

\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@{\string\bbl@scset\string\bbl@tempd}%
\ifin@% Renew caption
\bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
\ifin@
\bbl@exp% \\
\bbl@ifsamestring\{\bbl@tempa}\{\languagename\}%
\bbl@scset<#2name><#1#2name>}%
4.11 Macros common to a number of languages

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

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

4.12 Making glyphs available

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

4.12.1 Quotation marks

\quotedblbase In the T1 encoding the opening double quote at the baseline is available as a separate character, accessible via \quotedblbase. In the OT1 encoding it is not available, therefore we make it available by lowering the normal open quote character to the baseline.
Make sure that when an encoding other than OT1 or T1 is used this glyph can still be typeset.

\providecommanddefault\quotesinglbase{\usetextsymbol{OT1}{\quotesinglbase}}

We also need the single quote character at the baseline.

\providecommand{\quotesinglbase}{OT1}{\textforbidden{\quotesinglbase}}

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

\providecommanddefault\guillemetleft{\usetextsymbol{OT1}{\guillemetleft}}
\providecommanddefault\guillemetright{\usetextsymbol{OT1}{\guillemetright}}

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

\providecommand\guillemetleft{OT1}{\textforbidden{\guillemetleft}}
\providecommand\guillemetright{OT1}{\textforbidden{\guillemetright}}

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

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

\IJ

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

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

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

\IJ

\DJ

\dj

\DDJ

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

\begin{verbatim}
2158 \DeclareTextCommand{\SS}{OT1}{SS}
2159 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
\end{verbatim}

4.12.3 Shorthands for quotation marks

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

\begin{verbatim}
\glq \grq
\begin{verbatim}
2160 \ProvideTextCommandDefault{\glq}{% 
2161 \textornormalsize{\quotesinglbase}{\mbox{\quotesinglbase}}} 
\end{verbatim}
The definition of \grq depends on the font encoding. With T1 encoding no extra kerning is needed.

\begin{verbatim}
2162 \ProvideTextCommand{\grq}{T1}{% 
2163 \textornormalsize{\kern\z@	extquoteleft}{\mbox{\textquoteleft}}} 
2164 \ProvideTextCommand{\grq}{TU}{% 
2165 \textornormalsize{\textquoteleft}{\mbox{\textquoteleft}}} 
2166 \ProvideTextCommand{\grq}{OT1}{% 
2167 \save@sf@q{\kern-.0125em 
2168 \textornormalsize{\textquoteleft}{\mbox{\textquoteleft}}} 
2169 \kern.07em\relax} 
2170 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
\end{verbatim}
\end{verbatim}
\glqq \grqq
\begin{verbatim}
\begin{verbatim}
2171 \ProvideTextCommandDefault{\glqq}{% 
2172 \textornormalsize{\quotedblbase}{\mbox{\quotedblbase}}} 
\end{verbatim}
The definition of \grqq depends on the font encoding. With T1 encoding no extra kerning is needed.

\begin{verbatim}
2173 \ProvideTextCommand{\grqq}{T1}{% 
2174 \textornormalsize{\textquotedblleft}{\mbox{\textquotedblleft}}} 
2175 \ProvideTextCommand{\grqq}{TU}{% 
2176 \textornormalsize{\textquotedblleft}{\mbox{\textquotedblleft}}} 
2177 \ProvideTextCommand{\grqq}{OT1}{% 
2178 \save@sf@q{\kern-.07em 
2179 \textornormalsize{\textquotedblleft}{\mbox{\textquotedblleft}}} 
2180 \kern.07em\relax} 
2181 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
\end{verbatim}
\end{verbatim}
\flq \frq
\begin{verbatim}
\begin{verbatim}
2182 \ProvideTextCommandDefault{\flq}{% 
2183 \textornormalsize{\guilsinglleft}{\mbox{\guilsinglleft}}} 
2184 \ProvideTextCommandDefault{\frq}{% 
2185 \textornormalsize{\guilsinglright}{\mbox{\guilsinglright}}} 
\end{verbatim}
\end{verbatim}
\flqq \frqq
\begin{verbatim}
\begin{verbatim}
2186 \ProvideTextCommandDefault{\flqq}{% 
2187 \textornormalsize{\guillemetleft}{\mbox{\guillemetleft}}} 
2188 \ProvideTextCommandDefault{\frqq}{% 
2189 \textornormalsize{\guillemetteright}{\mbox{\guillemetteright}}} 
\end{verbatim}
\end{verbatim}
\end{verbatim}
\end{verbatim}
\end{verbatim}

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).

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

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

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

The following code fools \TeX's \texttt{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 \texttt{METAFONT} parameters with which the fonts were built. (Just try out, which value will look best.) If the new x-height is too low, it is not changed. Finally we call the \texttt{\accent} primitive, reset the old x-height and insert the base character in the argument.

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

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

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

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

\if\@english\undefined  
\chardef\@english\z@  
\fi

\% The following is used to cancel rules in ini files (see Amharic).
4.13 Layout

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

4.14 Load engine specific macros

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

Continue with \LaTeX\ only.

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

\begin{verbatim}
\texttt{% \package | core}
\texttt{\textbackslash \@error
\{This macro is available only in LuaLaTeX.\}}
\texttt{\{Consider switching to that engine.\}}
\texttt{\ifx\babelposthyphenation\@undefined
\let\babelposthyphenation\babelprehyphenation
\let\babelpatterns\babelprehyphenation
\let\babelcharproperty\babelprehyphenation
\fi

\section{Creating and modifying languages}

\texttt{4.15 Creating and modifying languages}

\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 ldf files.

\begin{verbatim}
\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 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.

\bbl@load@basic{#2}%
\% == script, language ==
\% Override the values from ini or defines them
\ifx\bbl@KVP@script@nnil\else
\bbl@csargedef{sname@#2}{\bbl@KVP@script}%
\fi
\ifx\bbl@KVP@language@nnil\else
\bbl@csargedef{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{%
\\AddToHook{env/document/before}{\\select@language(#2){}}%
\directlua{
if Babel.locale_mapped == nil then
Babel.locale_mapped = true
Babel.linebreaking.add_before(Babel.locale_map, 1)
Babel.loc_to_scr = {}
Babel.chr_to_loc = Babel.chr_to_loc or {}
end
Babel.locale_props[\the\localeid].letters = false
}%
\bbl@xin{ letters }{ \bbl@KVP@onchar\space}%
\ifin@
\directlua{
Babel.locale_props[\the\localeid].letters = true
}%
\fi
\bbl@xin{ ids }{ \bbl@KVP@onchar\space}%
\ifin@
\if\bbl@starthyphens@undefined % Needed if no explicit selection
\AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
\fi
\bbl@exp{\\bbl@add\\bbl@starthyphens
{\\bbl@patterns@lua{\languagename}}}%
\% TODO - error/warning if no script
\directlua{
if Babel.script_blocks[\bbl@cl{sbcp}] then
Babel.loc_to_scr[\the\localeid] =
Babel.script_blocks[\bbl@cl{sbcp}]
Babel.locale_props[\the\localeid].lc = \the\localeid\space
Babel.locale_props[\the\localeid].lg = \the@nameuse{l@\languagename}\space
end
}%
\fi
\bbl@xin{ fonts }{ \bbl@KVP@onchar\space}%
\ifin@
\bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}%
\bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{%
\directlua{
  if Babel.script_blocks[\'bbl@cl{sbcp}\'] then
    Babel.loc_to_scr[\the\localeid] =
    Babel.script_blocks[\'bbl@cl{sbcp}\']
  end}%
\ifx\bbl@mapselect\@undefined % TODO. almost the same as mapfont
  \AtBeginDocument{%
    \bbl@patchfont{{\bbl@mapselect}}%
    {\selectfont}%
    \def\bbl@mapselect{%
      \let\bbl@mapselect\relax
      \edef\bbl@prefontid{\fontid\font}%
      \def\bbl@mapdir##1{%
        {\def\languagename{##1}%
          \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
          \bbl@switchfont
          \directlua{Babel.locale_props[\the\csname bbl@id@@##1\endcsname\]
            \{\bbl@prefontid\} = \fontid\font}%
        }%}
    }%}
  }%\fi}
\bbl@exp{{{\bbl@add\www@bbl@mapselect{{\bbl@mapdir{\languagename}}}}}%}
\fi
% TODO - catch non-valid values
\fi
% == mapfont ==
% For bidi texts, to switch the font based on direction
\ifx\bbl@KVP@mapfont\@nnil\else
  \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
\fi
\bbl@provide@intraspace
% == Line breaking: intraspace, intrapenalty ==
% For CJK, East Asian, Southeast Asian, if interspace in ini
\ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
  \bbl@csarg\edef{intpsp@#2}{\bbl@KVP@intraspace}%
\fi
% == Line breaking: CJK quotes ==
\ifcase\bbl@engine
\bbl@xin{/c}{{\bbl@cl{lnbrk}}}%
\fi
\bbl@provide@intraspace
% == Line breaking: intraspace, intrapenalty ==
% For CJK, East Asian, Southeast Asian, if interspace in ini
\ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
  \bbl@csarg\edef{intpsp@#2}{\bbl@KVP@intraspace}%
\fi
% == Line breaking: CJK quotes ==
\ifcase\bbl@engine
\bbl@xin{/c}{{\bbl@cl{lnbrk}}}%
\fi
\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
end
  cs = ( cs == 'op') and 'cl' or 'op'
end
}}\fi
\fi
% == Line breaking: justification ==
\ifx\bbl@KVP@justification\@nnil\else
  \let\bbl@KVP@linebreaking=\bbl@KVP@justification
\fi
\ifx\bbl@KVP@linebreaking\@nnil\else
  \bbl@xin{,\bbl@KVP@linebreaking,}%
  {,elongated,kashida,cjk,padding,unhyphenated,}%
  \ifin@
    \bbl@csarg\xdef{lnbrk@\languagename}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
  \fi
  \bbl@xin{/e}{/}\bbl@cl{lnbrk}%
  \ifin@\bbl@arabicjust\fi
  \bbl@xin{/p}{/}\bbl@cl{lnbrk}%
  \ifin@\AtBeginDocument{\@nameuse{bbl@tibetanjust}}\fi
% == Line breaking: hyphenate.other.(locale|script) ==
\ifx\bbl@lbkflag\@empty
  \bbl@ifunset{bbl@hyotl@\languagename}{\%
    \bbl@csarg\replace{hyotl@\languagename}{ }{,}\%
    \bbl@startcommands*{\languagename}{}%
    \bbl@csarg\foreach{hyotl@\languagename}{%\ifcase\bbl@engine\ifnum##1<257
      \SetHyphenMap{\text{Babel Lower}(##1){##1}}\%
    \fi\else
      \SetHyphenMap{\text{Babel Lower}(##1){##1}}\%
    \fi}%\bbl@endcommands%
  }\%
  \bbl@csarg\foreach{hyots@\languagename}{%\ifcase\bbl@engine\global\lccode##1=##1\relax
    \fi\else
      \global\lccode##1=##1\relax
    \fi)%\bbl@endcommands%
% == Counters: maparabic ==
% Native digits, if provided in ini (TeX level, xe and lua)
\ifcase\bbl@engine\else
  \bbl@ifunset{bbl@dgnat@\languagename}{\%
    \bbl@setdigits{\csname bbl@dgnat@\languagename\endcsname}%
  }\%
  \bbl@startcommands*{\languagename}{}%
  \bbl@foreach{hyots@\languagename}{%\ifcase\bbl@engine\ifnum##1<257
    \global\lccode##1=##1\relax
  \fi
  \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
  \bbl@ifunset{bbl@dgnat@\languagename}{\%
    \bbl@setdigits{\csname bbl@dgnat@\languagename\endcsname}%
  }\%
  \bbl@startcommands*{\languagename}{}%
  \bbl@foreach{hyots@\languagename}{%\ifcase\bbl@engine\ifnum##1<257
    \global\lccode##1=##1\relax
  \fi
  \else
    \global\lccode##1=##1\relax
  \fi)%\bbl@endcommands%
\fi
Depending on whether or not the language exists (based on `\date<language>`), we define two macros. Remember `\bbl@startcommands` opens a group.

```latex
\def\bbl@provide@new#1{\@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
\@namedef{extras#1}{}% 
\@namedef{noextras#1}{}% 
\bbl@startcommands*{#1}{captions}% 
\ifx\bbl@KVP@captions\@nnil % and also if import, implicit 
\def\bbl@tempb##1{% elt for \bbl@captionslist
\expandafter\bbl@tempb\bbl@captionslist@empty
\else
\ifx\bbl@initoload\relax
\bbl@read@ini{%\bbl@KVP@captions}% % Here letters cat = 11
\else
\bbl@read@ini{%\bbl@initoload}% % Same
\fi
\fi
\StartBabelCommands*{#1}{date}% 
\ifx\bbl@KVP@date\@nnil
\bbl@exp[\SetString\##1{%#1}{\bbl@nocaption{today}{#1today}}]
\else
\bbl@savetoday
\bbl@savedate
```
Load the basic parameters (ids, typography, counters, and a few more), while captions and dates are left out. But it may happen some data has been loaded before automatically, so we first discard the saved values. (TODO. But preserving previous values would be useful.)

The hyphenrules option is handled with an auxiliary macro. This macro is called in three cases: when a language is first declared with \babelprovide, with hyphenrules and with import.
63
Now, the ‘main loop’, which **must be executed inside a group**. At this point, \bbl@inidata may contain data declared in \babelprovide, with ‘slashed’ keys. There are 3 steps: first read the ini file and store it; then traverse the stored values, and process some groups if required (date, captions, labels, counters); finally, ‘export’ some values by defining global macros (identification, typography, characters, numbers). The second argument is 0 when called to read the minimal data for fonts; with \babelprovide it's either 1 or 2.

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

\ifx\bbl@readstream@undefined
\csname newread\endcsname\bbl@readstream 
\fi
\def\bbl@read@ini#1#2{% 
\global\let\bbl@extend@ini\@gobble 
\openin\bbl@readstream=babel-#1.ini 
\ifeof\bbl@readstream 
\bbl@error 
{There is no ini file for the requested language\% (\languagename). Perhaps you misspelled it or your\% installation is not complete.)\% 
{Fix the name or reinstall babel.}\% 
\else 
% == Store ini data in \bbl@inidata == 
\catcode`\=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12 
\bbl@info{Importing \ifcase#2
\else
\fi
font and identification from babel-#1.ini. Reported}\% 
\global\catcode`\@=2 \z@
\global\let\bbl@inidata@empty 
\let\bbl@inistore@min \bbl@inistore % Remember it's local 
\fi 
\def\bbl@section{identification} 
\bbl@exp{\g@addto@macro\bbl@inidata{\bbl@section}{\bbl@tempa}{\the\toks@}}% 
\fi 
% == Process stored data == 
\bbl@exp{\\g@addto@macro\bbl@inidata{}{\bbl@section}{\bbl@tempa}{\the\toks@}}% 
% == 'Export' data == 
\bbl@ini@exports{#2} 
\global\let\bbl@inidata@empty 
}
A variant to be used when the ini file has been already loaded, because it's not the first \babelprovide for this language.

As a somewhat hackish tool to handle calendar sections. TODO. To be improved.
A key with a slash in \bbl@provide replaces the value in the ini file (which is ignored altogether). The mechanism is simple (but suboptimal): add the data to the ini one (at this point the ini file has not yet been read), and define a dummy macro. When the ini file is read, just skip the corresponding key and reset the macro (in \bbl@inistore above).

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

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 others sources do (asa private use extension). For consistency with other single-letter subtags or ‘singletons’, here is considered an extension, too.
\bbl@exportkey{lbcp}{identification.language.tag.bcp47}{%}
\bbl@exportkey{lotf}{identification.tag.opentype}{dflt}{%}
\bbl@exportkey{esname}{identification.script.name}{%}
\{\bbl@exportkey{sname}{identification.script.name.opentype}{%}
\bbl@exportkey{sbcp}{identification.script.tag.bcp47}{%}
\bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}{%}
\bbl@exportkey{rbcp}{identification.region.tag.bcp47}{%}
\bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{%}
\bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{%}
\bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{%}
\bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{%}
% Also maps bcp47 -> languagename
\ifbbl@bcptoname
\xdef{bcp@map@\bbl@cl{tbcp}}{\languagename}{%}
\fi
% Conditional
% 0 = only info, 1, 2 = basic, (re)new
\bbl@exportkey{calpr}{date.calendar.preferred}{%}
\bbl@exportkey{lnbrk}{typography.linebreaking}{h}{%}
\bbl@exportkey{hyphr}{typography.hyphenrules}{%}
\bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}{%}
\bbl@exportkey{rgthm}{typography.righthyphenmin}{3}{%}
\bbl@exportkey{prehc}{typography.prehyphenchar}{%}
\bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{%}
\bbl@exportkey{hyots}{typography.hyphenate.other.script}{%}
\bbl@exportkey{intsp}{typography.intraspace}{%}
\bbl@exportkey{frspc}{typography.frenchspacing}{u}{%}
\bbl@exportkey{chrng}{characters.ranges}{%}
\bbl@exportkey{quote}{characters.delimiters.quotes}{%}
\bbl@exportkey{dgnat}{numbers.digits.native}{%}
\ifnum#1=\tw@ % only (re)new
\bbl@error{The counter name 'digits' is reserved for mapping}\%
\{Use another name.\}{%}
\fi
\bbl@savecurrentvalues
\bbl@savetoday
\bbl@savedate
\bbl@savestrings
\fi
% A shared handler for key=val lines to be stored in \bbl@kv@<section>.<key>.
\def{\bbl@inikv@#1\#2}{% key=value
\toks@{#2} % This hides #'s from ini values
\bbl@csarg\edef{@kv@\bbl@section.#1}{\the	oks@}}
\let{\bbl@inikv@identification}{\bbl@inikv}
\let{\bbl@inikv@date}{\bbl@inikv}
\let{\bbl@inikv@typography}{\bbl@inikv}
\let{\bbl@inikv@characters}{\bbl@inikv}
\let{\bbl@inikv@numbers}{\bbl@inikv}

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

\let{\bbl@ini@identification}{\bbl@inikv}
\let{\bbl@ini@date}{\bbl@inikv}
\let{\bbl@ini@typography}{\bbl@inikv}
\let{\bbl@ini@characters}{\bbl@inikv}
\let{\bbl@ini@numbers}{\bbl@inikv}

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@ini@counters@\#1\#2}{%
\bbl@ifsamestring{\#1}{digits}\%
\{\bbl@error{The counter name 'digits' is reserved for mapping}\\%
\{decimal digits\}{%}
\{Use another name.\}{%}
\}
\def{\bbl@tempc}{\#1}{%
\bbl@trim@def{\bbl@tempb*}{\#2}{%}
\in@{.1\$}{\#1\$}{%}
\ifin@
\bbl@error{The counter name 'digits' is reserved for mapping}\\%
\{decimal digits\}{%}
\{Use another name.\}{%}
\}
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.</naturalsize
Labels. Captions must contain just strings, no format at all, so there is new group in ini files.

% The following code is still under study. You can test it and make suggestions. Eg, enumerate.2 = {enumerate}.1({enumerate}). It's language dependent.

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.

```latex
\def\bbl@chaptype{chapter}
\ifx@makechapterhead@undefined
\let\bbl@patchchapter\relax
\else\ifx\thechapter@undefined
\let\bbl@patchchapter\relax
\else\ifx\ps@headings@undefined
\let\bbl@patchchapter\relax
\else
\def\bbl@patchchapter{%
\global\let\bbl@patchchapter\relax
\gdef\bbl@chfmt{%
\bbl@ifunset{bbl@\bbl@chaptype fmt@\languagename}%
{\@chapapp\space thechapter}
{\@nameuse{bbl@\bbl@chaptype fmt@\languagename}}}
\bbl@add\appendix{\def\bbl@chaptype{appendix}}% Not harmful, I hope
\bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
\bbl@sreplace\@makechapterhead{\@chapapp\ \thechapter}{\bbl@chfmt}%
\bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
\bbl@toglobal\appendix
\bbl@toglobal\ps@headings
\bbl@toglobal\chaptermark
\bbl@toglobal@makechapterhead
\let\bbl@patchappendix\bbl@patchchapter
\fi\fi\fi
\ifx@part@undefined
\let\bbl@patchpart\relax
\else
\def\bbl@patchpart{%
\global\let\bbl@patchpart\relax
\gdef\bbl@partformat{%
\bbl@ifunset{bbl@partfmt@\languagename}%
{\partname\nobreakspace\thepart}
{\@nameuse{bbl@partfmt@\languagename}}}
\bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
\bbl@toglobal@part%
\fi
```

Date. Arguments (year, month, day) are not protected, on purpose. In `\today`, arguments are always
gregorian, and therefore always converted with other calendars. TODO. Document
Dates will require some macros for the basic formatting. They may be redefined by language, so "semi-public" names (camel case) are used. Oddly enough, the CLDR places particles like "de" inconsistently in either in the date or in the month name. Note after \bbl@replace \toks@ contains the resulting string, which is used by \bbl@replace@finish@iii (this implicit behavior doesn't seem a good idea, but it's efficient).
\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.}%
      \fi\fi\fi\fi}}%
\newcommand\BabelDateyyyy[1]{{\number#1}} % TODO - add leading 0
\def\bbl@replace@finish@iii#1{%
  \bbl@exp{\def\#1####1####2####3{%\the\toks@}}}
\def\bbl@TG@@date{%
  \bbl@replace\bbl@toreplace{[ \]}{\BabelDateSpace{}}%
  \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
  \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
  \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
  \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
  \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
  \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
  \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
  \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
  \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
  \bbl@replace\bbl@toreplace{[y|}{\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}

\transform{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{%
  \bbl@xin@{,,#1,}{,,#2,}{,,#3,}{,,#4,}{,,#5,}}%
\begingroup % A hack. TODO. Don't require an specific order%
  \catcode`%=12
  \catcode`&=14
  \gdef\bbl@transforms#1#2#3{&%
    \directlua{
      local str = [[#1]]
      str = str:gsub('%.%d+%.%d+$', '')
      token.set_macro('babeltempa', str)
    }&%
    \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
    \ifin@ &% font:font:transform syntax
      \directlua{
        local t = {}
        for m in string.gmatch('#1..',' ('..'')') do
          table.insert(t, m)
        end
        table.remove(t)
      }&%
    \fi
  \endif@}
  \bbl@foreach\bbl@KVP@transforms{&%
    \directlua{%
      local t = {}
      for m in string.gmatch('#1..',' ('..'')') do
        table.insert(t, m)
      end
      table.remove(t)
    }&%
  }&%
Language and Script values to be used when defining a font or setting the direction are set with the following macros.

```latex
\def\bbl@provide@lsys#1{\%
  \bbl@ifunset{bbl@lname@#1}{}\%
  {\bbl@load@info[#1]}\%
  \bbl@csarg\let{lsys@#1}\@empty\%
  \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}\%
  \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}\%
  \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}\%
  \bbl@ifunset{bbl@lname@#1}{}\%
  {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}\%
  \ifcase\bbl@engine\or\or\%
    \bbl@ifunset{bbl@prehc@#1}{}\%
    {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}{}\%
      \ifnum\hyphenchar\font=\defaulthyphenchar\%
        \iffontchar\font\bbl@cl{prehc}\relax\%
        \expandafter\@secondoftwo % to execute right now
        \fi
        \AtBeginDocument\%
        \bbl@patchfont{\bbl@xenohyph}\%
        \expandafter\select@language\expandafter{\languagename}\%
        \fi
      }\%
    }\%
  }\%
\bbl@csarg\bbl@toglobal{lsys@#1}}\%
\def\bbl@xenohyph@d{\%
  \bbl@ifset{bbl@prehc@\languagename}{}\%
  \iffontchar\font\bbl@cl{prehc}\relax\%
  \else\%
    \bbl@warning...\%
  \fi
\fi\%
\bbl@csarg\bbl@xenohyph@undefined\%
\global\let\bbl@xenohyph\bbl@xenohyph@d\%
\iffontchar\font\bbl@cl{prehc}\relax\%
  \expandafter\@secondoftwo % to execute right now
\fi
\AtBeginDocument\%
\bbl@warning...\%
```

Language and Script values to be used when defining a font or setting the direction are set with the following macros.
neither \texttt{0} nor \texttt{\textordmasculine{0}} \texttt{\textordmasculine{\textoumlaut{a}}} available in the current font, and therefore the hyphen will be printed. Try changing the fontspec's `HyphenChar' to another value, but be aware this setting is not safe (see the manual). Reported}

\begin{verbatim}
\hyphenchar\font\defaulthyphenchar \fi \fi \fi \fi \fi} \fi}
\end{verbatim}

\begin{verbatim}
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 \texttt{.tex} file named as the language (which means any code in it must be skipped, too).
\end{verbatim}

\begin{verbatim}
\def\bbl@load@info#1{%
\begin{verbatim}
\def\BabelBeforeIni##1##2{%
\begingroup
\bbl@read@ini{##1}0%
\endgroup % boxed, to avoid extra spaces:
{\bbl@input@texini{#1}}}
\end{verbatim}
\end{verbatim}

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.

\begin{verbatim}
\def\bbl@setdigits#1#2#3#4#5{%
\bbl@exp{%\def\languagename digits>####1{%
\let\bbl@digits@\languagename>####1\@nil}
\let\bbl@cntr@digits@\languagename>####1\expandafter\bbl@counter@\languagename>####1\csname c@####1\endcsname\def\bbl@counter@\languagename>####1{%
\expandafter\bbl@digits@\languagename>####1\number####1\@nil}}\def\bbl@tempa##1##2##3##4##5{%\bbl@exp{%\def\bbl@digits@\languagename>#####1{\ifx#####1\@nil % ie, \bbl@digits@lang\else\ifx0#####1#1\else\ifx1#####1#2\else\ifx2#####1#3\else\ifx3#####1#4\else\ifx4#####1#5\else\ifx5#####1#6\else\ifx6#####1#7\else\ifx7#####1#8\else\ifx8#####1#9\else#####1\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\f
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 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.
\LaTeX needs to know the BCP 47 codes for some features. For that, it expects BCP data to be defined. While language, region, script, and variant are recognized, extension (s) 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}\bbl@main@language}
  \{\bbl@bcpdata@ii{#1#2#3#4#5#6}\languagename\}}%
\def\bbl@bcpdata@ii#1#2{% 
  \bbl@ifunset{bbl@info@#1.tag.bcp47}{}{% \bbl@cs{bbl@info@#1.tag.bcp47}}}
\fi
% Still somewhat hackish. WIP.
\@namedef{bbl@info@casing.tag.bcp47}{casing}
\newcommand{\BabelUppercaseMapping}[3]{{ 
  \let\bbl@tempx\languagename
  \edef\languagename{#1}
  \DeclareUppercaseMapping{\BCPdata{casing}}{#2}{#3}
  \let\languagename\bbl@tempx}}
\newcommand{\BabelLowercaseMapping}[3]{{ 
  \let\bbl@tempx\languagename
  \edef\languagename{#1}
  \DeclareLowercaseMapping{\BCPdata{casing}}{#2}{#3}
  \let\languagename\bbl@tempx}}

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

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

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

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

\newcommand{\babeladjust}[1]{
\bbl@forkv{#1}{%
\bbl@ifunset{bbl@ADJ@##1@##2}{
\bbl@cs{ADJ@##1@##2}}%
{\bbl@cs{ADJ@##1}}%
\bbl@adjust@lua{#1}{##2}{##2}{##2}{##3}}%
\ifnum\currentgrouplevel=\z@
\directlua{ Babel.#2 }%
\expandafter\expandafter\expandafter\@gobble
\fi
\bbl@error % The error is gobbled if everything went ok.
{Currently, #1 related features can be adjusted only in the main vertical list.}%
{Maybe things change in the future, but this is what it is.}}%
\namedef{bbl@ADJ@bidi.mirroring@on}{\bbl@adjust@lua{bidi}{mirroring_enabled=true}}%
\namedef{bbl@ADJ@bidi.mirroring@off}{\bbl@adjust@lua{bidi}{mirroring_enabled=false}}%
\namedef{bbl@ADJ@bidi.text@on}{\bbl@adjust@lua{bidi}{bidi_enabled=true}}%
\namedef{bbl@ADJ@bidi.text@off}{\bbl@adjust@lua{bidi}{bidi_enabled=false}}%
\namedef{bbl@ADJ@bidi.math@on}{\let\bbl@noamsmath\@empty}%
\namedef{bbl@ADJ@bidi.math@off}{\let\bbl@noamsmath\relax}%
\namedef{bbl@ADJ@bidi.mapdigits@on}{\bbl@adjust@lua{bidi}{digits_mapped=true}}%
\namedef{bbl@ADJ@bidi.mapdigits@off}{\bbl@adjust@lua{bidi}{digits_mapped=false}}%
\namedef{bbl@ADJ@linebreak.sea@on}{\bbl@adjust@lua{linebreak}{sea_enabled=true}}%
\namedef{bbl@ADJ@linebreak.sea@off}{\bbl@adjust@lua{linebreak}{sea_enabled=false}}%
The \LaTeX \textbf{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.

5.1 Cross referencing macros

The \LaTeX\textbf{X} book states:

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

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

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

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

```latex
\def\bbl@savelastskip{%
  \let\bbl@restorelastskip\relax
  \ifvmode
    \ifdim\lastskip=z@\relax
    \let\bbl@restorelastskip\nobreak
    \else\bbl@exp{\def\bbl@restorelastskip{\skip@=\the\lastskip\nobreak\vskip-\skip@\vskip\skip@}}\fi\fi}}
\@namedef{bbl@ADJ@select.write@keep}{\let\bbl@restorelastskip\relax}
\@namedef{bbl@ADJ@select.write@omit}{\AddBabelHook{babel-select}{beforestart}{\expandafter\babel@aux\expandafter{\bbl@main@language}{}}\let\bbl@restorelastskip\relax}
\@namedef{bbl@ADJ@select.encoding@off}{\let\bbl@encoding@select@off\@empty}

\section{Cross referencing macros}

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.

\def\@testdef{\begin{quote}
\CheckCommand*{\@testdef}[3]{%#1\reserved@a{#3}}%#2\expandafter\ifx\csname#1@#2\endcsname\reserved@a%#2\else\latexwarning{There were multiply-defined labels}%#2\latexwarning{Label `#2' multiply defined}%#2\@global\@namedef{#1@#2}{#3}}%
\end{quote}}

\@newl@bel
An internal \LaTeX\textbf{X} macro used to test if the labels that have been written on the .aux file have changed. It is called by the \end{document} macro.

\def\reserved@a{#3}%
\else
\expandafter\ifx\csname csname@#1@#2\endcsname\reserved@a
\else
\end{document}
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 \@new@bel does it. When the label is defined we replace the definition of \bbl@tempa by its meaning. If the label didn't change, \bbl@tempa and \bbl@tempb should be identical macros.

\def\@testdef#1#2#3{% TODO. With @samestring?
  \@safe@activestrue
  \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
  \def\bbl@tempb{#3}%
  \@safe@activesfalse
  \ifx\bbl@tempa\relax
    \else
      \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
    \fi
  \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
  \ifx\bbl@tempa\bbl@tempb
    \else
      \@tempswatrue
    \fi
  \fi

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.

\bbl@xin@{R}\bbl@opt@safe
\ifin@
  \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
  \bbl@xin\{\expandafter\strip@prefix\meaning\ref}%
  \ifin@
    \bbl@redefine\@kernel@ref#1{%\@safe@activestrue\org@@kernel@ref{#1}\@safe@activesfalse}
    \bbl@redefine\@kernel@pageref#1{%\@safe@activestrue\org@@kernel@pageref{#1}\@safe@activesfalse}
    \bbl@redefine\@kernel@sref#1{%\@safe@activestrue\org@@kernel@sref{#1}\@safe@activesfalse}
    \bbl@redefine\@kernel@spageref#1{%\@safe@activestrue\org@@kernel@spageref{#1}\@safe@activesfalse}
  \else
    \bbl@redefinerobust\ref#1{%\@safe@activestrue\org@ref{#1}\@safe@activesfalse}
    \bbl@redefinerobust\pageref#1{%\@safe@activestrue\org@pageref{#1}\@safe@activesfalse}
  \fi
\else
  \let\org@ref\ref
  \let\org@pageref\pageref
\fi

\@citex
The macro used to cite from a bibliography, \cite, uses an internal macro, \@citex. It is this internal macro that picks up the argument(s), so we redefine this internal macro and leave \cite alone. The first argument is used for typesetting, so the shorthands need only be deactivated in the second argument.
Unfortunately, the packages natbib and cite need a different definition of \@citex... To begin with, natbib has a definition for \@citex with three arguments... We only know that a package is loaded when \begin{document} is executed, so we need to postpone the different redefinition.

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

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

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

\begin{verbatim}
\def\@citex[#1][#2][#3]{%
  \@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
  \org@citex[#1][#2]{\@tempa}}%
}\%
\end{verbatim}

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

\AtBeginDocument{\%}
\@ifpackageloaded{cite}{%}

\begin{verbatim}
\def\@citex[#1][#2][#3]{%
  \@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
  \org@citex[#1][#2]{\@tempa}}%
}\%
\end{verbatim}

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

\begin{verbatim}
\@bbl@redefine\nocite#1{%
  \@safe@activestrue\org@nocite{#1}\@safe@activesfalse}
\end{verbatim}

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

\begin{verbatim}
\@bbl@redefine\bibcite#1{%
  \@safe@activestrue\org@bibcite{#1}\@safe@activesfalse}
\end{verbatim}

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

\begin{verbatim}
\def\bbl@bibcite#1#2{%
  \@safe@activestrue\org@bibcite{#1}[\@safe@activesfalse#2]}
\end{verbatim}

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

\begin{verbatim}
\def\bbl@cite@choice{%
  \@global\let\bbl@bibcite\bbl@bibcite
  \@ifpackageloaded{natbib}{\@global\let\bbl@bibcite\org@bibcite}\%
  \@ifpackageloaded{cite}{\@global\let\bbl@bibcite\org@bibcite}\%
  \@global\let\bbl@choice\relax}
\end{verbatim}

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

\AtBeginDocument{\%}
\bbl@cite@choice

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

\begin{verbatim}
\@bbl@redefine\@bibitem#1{%
  \@safe@activestrue\org@bibitem{#1}\@safe@activesfalse}
\end{verbatim}

81
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.

3801 \bbl@trace{Marks}
3802 \IfBabelLayout{sectioning}{
\ifx\bbl@opt@headfoot@mnil
\@addto@macro\@resetactivechars{%
\set@typeset@protect
\expandafter\select@language@x\expandafter{\bbl@main@language}%
\let\protect\noexpand
\ifcase\bbl@bidimode\else % Only with bidi. See also above
\edef\thepage{%
\noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
\noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}}%
\fi
\fi}

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

3822 \iffalse\@mkboth\markboth
3823 \def\bbl@tempc{\let\@mkboth\markboth}\
3824 \else
3825 \def\bbl@tempc{}\
3826 \fi
3827 \bbl@ifunset{markboth }\bbl@redefine\bbl@redefinerobust
3828 \markboth#1#2{\
\protected@edef\bbl@tempb##1{\
\protect\foreignlanguage\{\languagename}{\protect\bbl@restore@actives##1}}\
\bbl@ifblank{#1}{}\
\toks@{#1}\
\bbl@exp{\
\org@markright{\protect\foreignlanguage\{\languagename}{\protect\markboth{#1}}}}\
\bbl@tempc
\fi

5.3 Preventing clashes with other packages

5.3.1 ifthen

\ifthenelse Sometimes a document writer wants to create a special effect depending on the page a certain
fragment of text appears on. This can be achieved by the following piece of code:
In order for this to work the argument of \isodd needs to be fully expandable. With the above
redefinition of \pageref it is not in the case of this example. To overcome that, we add some code to
the definition of \ifthenelse to make things work.
We want to revert the definition of \pageref and \ref to their original definition for the first
argument of \ifthenelse, so we first need to store their current meanings.
Then we can set the @safe@actives switch and call the original \ifthenelse. In order to be able to
use shorthands in the second and third arguments of \ifthenelse the resetting of the switch and the
definition of \pageref happens inside those arguments.

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

5.3.2 \texttt{varioref}

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

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

The package varioref defines \Ref to be a robust command which upperscases 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 \texttt{\Ref}, to call \texttt{\org@ref} instead of
\texttt{\ref}. The disadvantage of this solution is that whenever the definition of \texttt{\Ref} changes, this
definition needs to be updated as well.

\begin{verbatim}
\expandafter\def\csname Ref \endcsname#1{%  
\protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
\end{verbatim}
5.3.3 \texttt{hhline}

\texttt{hhline}

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

\begin{verbatim}
5.3.3 hhline
\fi

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

\begin{verbatim}
5.3.3 hhline
\fi

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

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

\texttt{Deprecate. Use the tools provide by \LaTeXX. The command \texttt{\substitutefontfamilyfamily} creates an \texttt{.fd} file on the fly. The first argument is an encoding mnemonic, the second and third arguments are font family names.}

\begin{verbatim}
5.3.3 hhline
\fi

\texttt{Deprecate. Use the tools provide by \LaTeXX. The command \texttt{\substitutefontfamilyfamily} creates an \texttt{.fd} file on the fly. The first argument is an encoding mnemonic, the second and third arguments are font family names.}
\end{verbatim}

\begin{verbatim}
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 \texttt{@fontenc@load@list}. If a non-ASCII has been loaded, we define versions of \texttt{\LaTeX} and \texttt{\LaTeX} for them using \texttt{\ensureascii}. The default ASCII encoding is set, too (in reverse order): the `main' encoding (when the document begins), the last loaded, or \texttt{OT1}.

\texttt{\ensureascii}
\end{verbatim}

\begin{verbatim}
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 \texttt{@fontenc@load@list}. If a non-ASCII has been loaded, we define versions of \texttt{\LaTeX} and \texttt{\LaTeX} for them using \texttt{\ensureascii}. The default ASCII encoding is set, too (in reverse order): the `main' encoding (when the document begins), the last loaded, or \texttt{OT1}.

\texttt{\ensureascii}
\end{verbatim}
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}%
    \ifx\UTFencname@undefined
      EU\ifcase\bbl@engine\or2\or1\fi
    \else
      \UTFencname
    \fi%
  \else
    \UTFencname%
  \fi}%
\end{verbatim}

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

\DeclareRobustCommand{\latintext}{%
  \fontencoding{\latinencoding}\selectfont
  \def\encodingdefault{\latinencoding}}
\textlatin This command takes an argument which is then typeset using the requested font encoding. In order to avoid many encoding switches it operates in a local scope.

\verbatim
\DeclareTextFontCommand \textlatin \leavevmode \latintext #1
\else
\DeclareTextFontCommand \textlatin \latintext
\fi

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

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

5.5 Basic bidi support

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

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

There are two ways of modifying macros to make them “bidi”, namely, by patching the internal low-level macros (which is what I have done with lists, columns, counters, tocs, much like rl babel did), and by introducing a “middle layer” just below the user interface (sectioning, footnotes).

- \texttt{pdftex} provides a minimal support for bidi text, and it must be done by hand. Vertical typesetting is not possible.
- \texttt{xetex} is somewhat better, thanks to its font engine (even if not always reliable) and a few additional tools. However, very little is done at the paragraph level. Another challenging problem is text direction does not honour \TeX\ grouping.
- \texttt{luatex} can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but bidi text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As \texttt{Lua\TeX}\-ja shows, vertical typesetting is possible, too.

\verbatim
\bbl@trace{Loading basic (internal) bidi support}
\ifodd\bbl@engine
\else % TODO. Move to tt babel
\ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200 % Any xe+lua bidi=
\bbl@error
{The bidi method 'basic' is available only in Lua\TeX. I'll continue with 'bidi=default', so expect wrong results}
{See the manual for further details}
\let\bbl@beforeforeign\leavevmode
\AtEndOfPackage{\EnableBabelHook{babel-bidi}}
\bbl@xebidipar
\fi\fi
\def\bbl@loadxebidi#1{\ifx\RTLfootnotetext\@undefined
\AtEndOfPackage{\EnableBabelHook{babel-bidi}}
\usepackage#1{bidi}}\fi}
\ifnum\bbl@bidimode>200 % Any xe bidi=
\ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
\bbl@tentative{bidi=bidi}
\bbl@loadxebidi{}
or
\bbl@loadxebidi{[rldocument]}
or
\bbl@loadxebidi{}
\fi
\fi\fi
\verbatim
Now come the macros used to set the direction when a language is switched. First the (mostly) common macros.

\\def\bbl@alscripts\{Arabic,Syriac,Thaana,\}
\def\bbl@rscripts{% TODO. Base on codes ??
,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
Old Hungarian,Lydian,Mandaean,Manichaean,%
Meroitic Cursive,Meroitic,Old North Arabian,%
Nabataean,N’Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
Old South Arabian,}%
\def\bbl@provide@dirs#1{% TODO - math
\\bbl@xin@\{\csname bbl@sname@#1\endcsname\}\{\bbl@alscripts\bbl@rscripts\}%
\ifin@
\global\bbl@csarg\chardef\wdir@#1\@ne
\\bbl@xin@\{\csname bbl@sname@#1\endcsname\}\{\bbl@alscripts\}%
\ifin@
\global\bbl@csarg\chardef\wdir@#1\tw@ % useless in xetex
\else
\global\bbl@csarg\chardef\wdir@#1\z@ %
\fi
\fi
\ifodd\bbl@engine
\bbl@csarg\textcase\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
\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}\% <- Must precede \bbl@textdir
\bbl@textdir\#1}%
\bbl@switchdir\%
\bbl@ifunset{bbl@lvsys\{\languagename\}}{%}
\bbl@exp{\bbl@setdirs\bbl@cl\wdir}\%
\bbl@switchdir\% TDOO - math
\textcase\bbl@select@type\% TDOO - strictly, not the right test
\bbl@bodydir\#1% <- Must precede \bbl@textdir
\bbl@textdir\#1}%
\bbl@switchdir\%
\textcase\bbl@select@type\% TDOO. 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@
    \bbl@textdir@i\beginL\endL
  \else
    \chardef\bbl@thetextdir\@ne
    \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}%
        \bgroup\aftergroup#2\aftergroup\egroup
      \else
        \ifcase\currentgrouptype\or % 0 bottom
          \aftergroup#2% 1 simple {}
        \or
          \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
        \or
          \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
        \or
          \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
        \or
          \bgroup\aftergroup#2\aftergroup\egroup % output math disc insert vcent mathchoice
        \or
          \bgroup\aftergroup#2\aftergroup\egroup % 14 \begingroup
        \else
          \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
        \fi
      \fi
      \fi
    \fi
    \bbl@dirlevel\currentgrouplevel
  \fi
  #1%
  \fi}
\def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
\let\bbl@bodydir\@gobble
\let\bbl@pagedir\@gobble
\def\bbl@dirparastext{\chardef\bbl@thepardir\bbl@thetextdir}
\everypar hack for xetex, to properly handle the par direction. Note text and par dirs are decoupled to some extent (although not completely).
\let\bbl@xebidipar\relax
\ifcase\bbl@thepardir\relax
  \let\bbl@xebidipar\relax
  \TeXXeTstate\me
\else
  \def\bbl@xeeverypar{%
    \ifcase\bbl@thepardir
      \ifcase\bbl@thetextdir\else\beginR\fi
    \else
      \setbox\z@\lastbox\beginR\box\z@%
    \fi
  \fi
  \let\bbl@seeverypar\everypar
  \newtoks\everypar
  \everypar=\bbl@seeverypar\everypar
  \bbl@seeverypar{\bbl@xeeverypar\the\everypar}
A tool for weak L (mainly digits). We also disable warnings with hyperref.

\AtBeginDocument{\ifx\pdfstringdefDisableCommands\@undefined\else\pdfstringdefDisableCommands{\let\babelsublr\@firstofone}\fi}

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

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

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

\input works, too, but possible errors are not caught.

\loadlocalcfg
\AtBeginDocument{\ifx\pdfstringdefDisableCommands\@undefined\else\pdfstringdefDisableCommands{\let\babelsublr\@firstofone}\fi}

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

\loadlocalcfg

\AtBeginDocument{\ifx\pdfstringdefDisableCommands\@undefined\else\pdfstringdefDisableCommands{\let\babelsublr\@firstofone}\fi}

\declarelanguage{weakL}{\babelsublr[1]}\setmainlanguage{weakL}

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).
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.}}

Now, we set a few language options whose names are different from ldf files. These declarations are preserved for backwards compatibility, but they must be eventually removed. Use proxy files instead.

\def\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}{}}

Another way to extend the list of 'known' options for babel was to create the file \bbllopts.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.

\ifx\bbl@opt@config\@nnil\ifpackagewith{babel}{noconfigs}{}%\InputIfFileExists{bblopts.cfg}{{\typeout{*************************************\^\^J* Local config file bblopts.cfg used\^\^J*}}}{\bbl@error{{Local config file '\bbl@opt@config.cfg' not found}\% Perhaps you misspelled it.}}\else\InputIfFileExists{\bbl@opt@config.cfg}{{\typeout{*************************************\^\^J* Local config file \bbl@opt@config.cfg used\^\^J*}}}{\bbl@error{}}\fi\fi

Recognizing global options in packages not having a closed set of them is not trivial, as for them to be processed they must be defined explicitly. So, package options not yet taken into account and stored in \bbl@language@opts are assumed to be languages. If not declared above, the names of the option and the file are the same. We first pre-process the class and package options to determine the main language, which is processed in the third 'main' pass, except if all files are ldf and there is no main key. In the latter case (\bbl@opt@main is still \@nnil), the traditional way to set the main language is kept — the last loaded is the main language.

\ifx\bbl@opt@main\@nnil\ifnum\bbl@inilflag\@0 % if all ldf's: set implicitly, no main pass \let\bbl@tempb\empty \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}\bbl@foreach\bbl@tempa{% \bbl@tempb is a reversed list \ifx\bbl@opt@main\@nnil % ie, if not yet assigned \ifodd\bbl@iniflag % *=

90
IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}
\else % n +=
\IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}
\fi
\fi}
\fi}
\else
\bbl@info{Main language set with 'main='. Except if you have\%
problems, prefer the default mechanism for setting\%
the main language, ie, as the last declared.\%
Reported}
\fi

A few languages are still defined explicitly. They are stored in case they are needed in the ‘main’ pass
(the value can be \relax).
\if\bbl@opt@main@nil\else
\bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}\
\expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
\fi

Now define the corresponding loaders. With package options, assume the language exists. With class
options, check if the option is a language by checking if the correspondind file exists.
\bbl@foreach\bbl@language@opts{%
\def\bbl@tempa{#1}%
\if\bbl@tempa\bbl@opt@main\else
\if\num\bbl@iniflag<\tw@ % 0 ø (other = ldf)
\bbl@ifunset{ds@#1}%
{\DeclareOption{#1}{\bbl@load@language{#1}}}%
}%
\else % + * (other = ini)
\DeclareOption{#1}{%
\bbl@ldfinit
\babelprovide[import]{#1}%
\bbl@afterldf{}}%
\fi
\fi}
\bbl@foreach\@classoptionslist{%
\def\bbl@tempa{#1}%
\if\bbl@tempa\bbl@opt@main\else
\if\num\bbl@iniflag<\tw@ % 0 ø (other = ldf)
\bbl@ifunset{ds@#1}%;
{\IfFileExists{#1.ldf}%
{\DeclareOption{#1}{\bbl@load@language{#1}}}%
}{}
\else % + * (other = ini)
\IfFileExists{babel-#1.tex}{
{\DeclareOption{#1}{\bbl@load@language{#1}}}%
}{}
\fi
\fi}

And we are done, because all options for this pass has been declared. Those already processed in the
first pass are just ignored.
The options have to be processed in the order in which the user specified them (but remember class
options are processes before):
\def\AfterBabelLanguage#1{%
\bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}%
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 \bblprovide can’t go inside a \DeclareOption; this explains why it’s executed directly, with a dummy declaration. Then all languages have been loaded, so we deactivate \AfterBabelLanguage.

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.

\def\bbl@usehooks@lang{/}{begindocument}{()}}\}
\def\AfterBabelLanguage{\}
\bbl@error{Too late for \string\AfterBabelLanguage{Languages have been loaded, so I can do nothing})

\ifx\bbl@main@language@undefined\bbl@info{You haven’t specified a language as a class or package\%
\bbl@load@language{nil}
The kernel of Babel (\texttt{babel.def}, common)

The kernel of the babel system is currently stored in \texttt{babel.def}. The file \texttt{babel.def} contains most of the code. The file \texttt{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 \texttt{hyphen.cfg} but \texttt{etex.src}, which follows a different naming convention, so we need to define the babel names. It presumes \texttt{language.def} exists and it is the same file used when formats were created.

A proxy file for switch.def

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

Loading hyphenation patterns

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

\begin{verbatim}
\ProvidesFile{hyphen.cfg}[\langle\langle date\rangle\rangle v\langle\langle version\rangle\rangle Babel hyphens]
\xdef\bbl@format{\jobname}
\def\bbl@version{\langle\langle version\rangle\rangle}
\def\bbl@date{\langle\langle date\rangle\rangle}
\ifx\AtBeginDocument\@undefined
\def\@empty{}\fi
\end{verbatim}

\process@line

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

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

\process@synonym

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

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

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

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

\begin{verbatim}
\def\process@synonym#1{%
\ifnum\last@language=m@ne
\toks@expandafter{\the\toks@\relax}\process@synonym{#1}%
\else
\expandafter\chdef\csname l@#1\endcsname\last@language
\end{verbatim}
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 language\rangle/hyphenmins macro. When no assignments were made we provide a default setting.

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

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

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

Finally, if the counter \language is equal to zero we execute the synonyms stored.
\bbl@get@enc The macro \bbl@get@enc extracts the font encoding from the language name and stores it in \bbl@hyph@enc. It uses delimited arguments to achieve this.

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

\bbl@hook@everylanguage#1{}
\bbl@hook@loadpatterns#1\input #1\relax
\let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
\def\addlanguage{\csname newlanguage\endcsname}
\def\adddialect##1##2{\global\chardef##1##2\relax
\wlog{\string##1 = a dialect from \string\language##2}}
\def\iflanguage##1{\expandafter\ifx\csname l@##1\endcsname\relax
\@nolanerr{##1}\else\ifnum\csname l@##1\endcsname=\language\expandafter\expandafter\expandafter\@firstoftwo\else\expandafter\expandafter\expandafter\@secondoftwo\fi\fi}
\def\providehyphenmins##1##2{\expandafter\ifx\csname ##1hyphenmins\endcsname\relax
\@namedef{##1hyphenmins}{##2}\fi}
\def\set@hyphenmins##1##2{\lefthyphenmin##1\relax\righthyphenmin##2\relax}
\def\selectlanguage{\errhelp{Selecting a language requires a package supporting it}\errmessage{Not loaded}}
\let\foreignlanguage\selectlanguage\let\otherlanguage\selectlanguage\let\otherlanguage*\selectlanguage
\let\localename\setlocale\let\textlocale\setlocale\let\textlanguage\setlocale\let\languagetext\setlocale
\begingroup
\def\AddBabelHook#1#2{\expandafter\ifx\csname bbl@hook@#2\endcsname\relax
\def\next{\toks1}\else\def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}\fi\fi
\def\set@hyphenmins##1##2{\lefthyphenmin##1\relax\righthyphenmin##2\relax\def\selectlanguage{\errhelp{Selecting a language requires a package supporting it}\errmessage{Not loaded}}\let\foreignlanguage\selectlanguage\let\otherlanguage\selectlanguage\let\otherlanguage*\selectlanguage\expandafter\expandafter\expandafter\@firstoftwo\expandafter\expandafter\expandafter\@secondoftwo\fi\fi
\def\setlocale{\errhelp{Find an armchair, sit down and wait}\errmessage{Not yet available}}\let\uselocale\setlocale\let\setlocale\setlocale
\let\selectlocale\setlocale\let\localename\setlocale\let\textlocale\setlocale\let\textlanguage\setlocale\let\languagetext\setlocale
\begingroup
\def\AddBabelHook#1#2{\expandafter\ifx\csname bbl@hook@#2\endcsname\relax
\def\next{\toks1}\else\def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}\fi\fi
\endgroup
\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
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\m@ne
\read1 to \bbl@line
\endlinechar`\^^M
If the file has reached its end, exit from the loop here. If not, empty lines are skipped. Add 3 space characters to the end of \bbl@line. This is needed to be able to recognize the arguments of \process@line later on. The default language should be the very first one.
\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
 \def\languagename{#1}%
 \def\bbl@elt##1##2##3##4{}}%
\bbl@languages
\endgroup
\fi
\closein1
We add a message about the fact that babel is loaded in the format and with which language patterns to the everyjob register.

```
4481 \if\the\toks@\else
4482 \errhelp{language.dat loads no language, only synonyms}
4483 \errmessage{Orphan language synonym}
4484 \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.

```
4485 \let\bbl@line\@undefined
4486 \let\process@line\@undefined
4487 \let\process@synonym\@undefined
4488 \let\process@language\@undefined
4489 \let\bbl@get@enc\@undefined
4490 \let\bbl@hyph@enc\@undefined
4491 \let\bbl@tempa\@undefined
4492 \let\bbl@hook@loadkernel\@undefined
4493 \let\bbl@hook@everylanguage\@undefined
4494 \let\bbl@hook@loadpatterns\@undefined
4495 \let\bbl@hook@loadexceptions\@undefined
```

Here the code for \LaTeX{} ends.

## 8 Font handling with fontspec

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

```
4497 ⟨⟨More package options⟩⟩ ≡
4498 \chardef\bbl@bidimode\z@
4499 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4500 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4501 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4502 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4503 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4504 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
```

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.

```
4506 ⟨⟨Font selection⟩⟩ ≡
4507 \bbl@trace{Font handling with fontspec}
4508 \ifx\ExplSyntaxOn\@undefined\else
4509 \def\bbl@fs@warn@nx#1#2{% \bbl@tempfs is the original macro
4510 \in@{,#1,}{,no-script,language-not-exist,}%
4511 \ifin@\else\bbl@tempfs@nx#1#2\fi}
4512 \def\bbl@fs@warn@nxx#1#2#3{%
4513 \in@{,#1,}{,no-script,language-not-exist,}%
4514 \ifin@\else\bbl@tempfs@nxx#1#2#3\fi}
4515 \def\bbl@loadfontspec{%
4516 \let\bbl@loadfontspec\relax
4517 \ifx\fontspec\@undefined
4518 \usepackage{fontspec}%
4519 \fi%
4520 \fi
4521 \onlypreamble\babelfont
4522 \newcommand\babelfont[2][]{% 1=langs/scripts 2=fam
4523 \bbl@foreach{#1}{%
\expandafter\ifx\csname date\endcsname\relax
\IfFileExists{babel-\endcsname\relax}{\babelprovide{\endcsname\relax}{}}\fi
\edef\bbl@tempa{#1}%
def\bbl@tempb{#2}% Used by \bbl@bblfont
\bbl@loadfontspec
\EnableBabelHook{babel-fontspec} Just calls \bbl@switchfont
\bbl@bblfont
\newcommand\bbl@bblfont[2]{% 1=features 2=fontname, @font=rm|sf|tt
\bbl@if_unset{\bbl@tempb family}{}{bl@providefam{\bbl@tempb}}%
% For the default font, just in case:
\bbl@if_unset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}
\expandafter\bbl@ifblank{\bbl@tempa}{% ie bbl@rmdflt@lang / *scrt
\bbl@foreach\bbl@tempa{%...

If the family in the previous command does not exist, it must be defined. Here is how:
\def\bbl@providefam{#1}{%
\bbl@exp{%...}

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@nostaftfont{#1}{%
\bbl@exp{%...}

If you consider defining a new family with \string\babelfont. See the manual for further details about \string\babelfont. Reported}}

{}}%
gdef\bbl@switchfont{
\bbl@exp{%...}

\bbl@exp{%...}

123=F - nothing!
3=T - from generic

1=F - from script?
2=F - from generic

(1) language?
(2) from script?
(3) from generic

There is nothing intrinsically wrong with this warning, and you can ignore it altogether if you do not need these families. But if they are used in the document, you should be aware 'babel' will not set Script and Language for them, so you may consider defining a new family with \string\babelfont.
The following is executed at the beginning of the aux file or the document to warn about fonts not defined with \babelfont.

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.
9 Hooks for XeTeX and LuaTeX

9.1 XeTeX

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

\def\BabelFootnote#1#2#3#4{\ifx\bbl@fn@footnote\@undefined\let\bbl@fn@footnote\footnote\fi\ifx\bbl@fn@footnotetext\@undefined\let\bbl@fn@footnotetext\footnotetext\fi\bbl@ifblank{#2}{\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}{\def#1{\bbl@exp{\\\bbl@footnote{\\\foreignlanguage{#2}}}{#3}{#4}}}\@namedef{\bbl@stripslash#1text}{\bbl@footnotetext{\\\foreignlanguage{#2}}}{#3}{#4}}}
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. Consider txt babel as a shorthand for \texttt{tex-xet babel}, which is the bidi model in both pdftex and xetex.

[Page 102]
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.

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.cfm (without the hyphenmins stuff, which is under the direct control of babel).

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

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

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

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

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

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

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

\begin{verbatim}
4913 \langle\langleluatex\rangle
4914 \ife\texttt{AddBabelHook}\texttt{\@undefined} % When plain.def, babel.sty starts
4915 \bbl@trace{Read language.dat}
4916 \ife\bbl@readstream\texttt{\@undefined}
4917 \csname newread\endcsname\bbl@readstream
4918 \fi
4919 \begingroup
4920 \toks@{}
4921 \count@\z@ % 0=start, 1=0th, 2=normal
4922 \def\bbl@process@line#1#2 #3 #4 {%
4923 \ifx=#1%
4924 \bbl@process@synonym{#2}%
4925 \else
4926 \bbl@process@language{#1#2}{#3}{#4}%
4927 \fi
4928 \ignorespaces}
4929 \def\bbl@manylang{%
4930 \ifnum\bbl@last>\@ne
4931 \bbl@info{Non-standard hyphenation setup}%
4932 \fi
4933 \let\bbl@manylang\relax}
4934 \def\bbl@process@language#1#2#3{%
4935 \ifcase\count@
4936 \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}(%
4937 \or
4938 \count@\tw@%
4939 \fi
4940 \ifnum\count@\@tw@
4941 \expandafter\addlanguage\csname l@#1\endcsname
4942 \language\allocationnumber
4943 \chardef\bbl@allocationnumber
4944 \bbl@manyLang
4945 \let\bbl@elt\relax
4946 \xdef\bbl@languages{%
4947 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}%)%
4948 \fi
4949 \the\toks@
4950 \toks@{}}
\end{verbatim}

105
\def\bbl@process@synonym@aux#1#2{\global\expandafter\chardef\csname l@#1\endcsname#2\relax\let\bbl@elt\relax\xdef\bbl@languages{\bbl@languages\bbl@elt{#1}{#2}{}{}}}\def\bbl@process@synonym#1{\ifcase\count@\toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}\or\@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}\else\bbl@process@synonym@aux{#1}{\the\bbl@last}\fi}
\ifx\bbl@languages\@undefined % Just a (sensible?) guess\chardef\l@english\z@
\chardef\l@USenglish\z@
\chardef\bbl@last\z@
\global\@namedef{bbl@hyphendata@0}{{hyphen.tex}{}%\gdef\bbl@languages{\bbl@elt{english}{0}{hyphen.tex}{}\bbl@elt{USenglish}{0}{}{}\gdef\bbl@languages@format\bbl@languages\gdef\bbl@elt#1#2#3#4{% Remove all except language 0\ifnum#2>\z@\else\noexpand\bbl@elt{#1}{#2}{#3}{#4}\fi}\xdef\bbl@languages{\bbl@languages}\fi\def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags\bbl@languages\openin\bbl@readstream=language.dat\ifeof\bbl@readstream\bbl@warning{I couldn’t find language.dat. No additional patterns loaded. Reported}%\else\loop\endlinechar\m@ne\read\bbl@readstream to \bbl@line\endlinechar“\^^M\if\T\ifeof\bbl@readstream F\fi \T\relax\ifeof\bbl@readstream F\fi \relax\edef\bbl@line{\bbl@line\space\space\space}%\expandafter\bbl@process@line\bbl@line\relax\fi\repeat\fi\closein\bbl@readstream\endgroup\bbl@trace{Macros for reading patterns files}\def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}\ifx\babelcatcodetablenum\@undefined\ifx\newcatcodetable\@undefined\def\babelcatcodetablenum{5211}\def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}\else\newcatcodetable\babelcatcodetablenum\newcatcodetable\bbl@pattcodes\fi\else\def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}\fi\else\def\bbl@pattcodes{\numexpr\babelcatcodetablelenum+1\relax}\fi\def\bbl@luapatterns#1#2{\ifx\bbl@languages\undefined}
\begin{verbatim}
Babel = Babel or {}
function Babel.bytes(line)
  return line:gsub("(.)", function (chr) return unicode.utf8.char(string.byte(chr)) end)
end
function Babel.begin_process_input()
  if luatexbase and luatexbase.add_to_callback then
    luatexbase.add_to_callback('process_input_buffer', Babel.bytes, 'Babel.bytes')
  else
    Babel.callback = callback.find('process_input_buffer')
    callback.register('process_input_buffer',Babel.bytes)
  end
end
function Babel.end_process_input ()
  if luatexbase and luatexbase.remove_from_callback then
    luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
  else
    callback.register('process_input_buffer',Babel.callback)
  end
end
function Babel.addpatterns(pp, lg)
  local lg = lang.new(lg)
  local pats = lang.patterns(lg) or ''
  lang.clear_patterns(lg)
  for p in pp:gmatch('[^%s]+') do
    ss = ''
    for i in string.utfcharacters(p:gsub('%d', '')) do
      ss = ss .. '%d?..' .. i
    end
    ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
    ss = ss:gsub('%.%%d%?$', '%%."
    pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
    if n == 0 then
      tex.sprint(
        "[[\string\csname\space bbl@info\endcsname{New pattern: ]
        .. 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.d or nil
      if dir == 'K' or dir == 'I' then
        has_bidi = true
      end
    end
  end
  return has_bidi
end
\end{verbatim}
if not dir then
  for nn, et in ipairs(ranges) do
    if itemchar < et[1] then
      break
    elseif itemchar <= et[2] then
      dir = et[3]
      break
    end
  end
  if dir and (dir == 'al' or dir == 'r') then
    has bidi = true
  end
end
return has bidi
end

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

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

}\endgroup
\ifx\newattribute\@undefined\else
  \newattribute\bbl@attr@locale
  \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
  \AddBabelHook{luatex}{beforeextras}{%}
    \setattribute\bbl@attr@locale\localeid}
\fi
\def\BabelStringsDefault{unicode}
\let\luabbl@stop\relax
\AddBabelHook{luatex}{encodedcommands}{%
  \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
  \ifx\bbl@tempa\bbl@tempb\else%
    \directlua{Babel.begin_process_input()}%
  \def\luabbl@stop{%
    \directlua{Babel.end_process_input()}}%
  \fi}%
  \AddBabelHook{luatex}{stopcommands}{%
  \luabbl@stop%
  \let\luabbl@stop\relax}
\AddBabelHook{luatex}{patterns}{%
  \ifundefined{bbl@hyphendata@\the\language}{}
  \ifnum\#2=\cname l@#2\endcsname % #2=spanish, dutch:OT1...
    \def\bbl@tempa{\#3\#4}%
  \ifx\bbl@tempa\empty\else % if not a synonymous
    \def\bbl@tempc{(#3)(#4)}%
  \fi
  \\bbl@csarg\xdef{hyphendata@#2}{{\bbl@tempc}}%
  \bbl@csarg
  \else
    \bbl@csarg\xdef{hyphendata@#2}{{\bbl@tempc}}%
  \fi
  \fi
}
\babelpatterns This macro adds patterns. Two macros are used to store them: `\bbl@patterns@` for the global ones and `\bbl@patterns@<lang>` for language ones. We make sure there is a space between words when multiple commands are used.

\begin{verbatim}
\onlypreamble\babelpatterns
\AtEndOfPackage{%
\newcommand\babelpatterns[2][\empty]{%\empty}
\ifx\bbl@patterns@\relax
\let\bbl@patterns@\empty
\fi
\ifx\bbl@pttnlist\empty\else
\bbl@warning{You must not intermingle `\selectlanguage\space` and `\babelpatterns\space` or some patterns will not be taken into account. Reported}%
\string\babelpatterns\space or some patterns will not be taken into account. Reported}%
\fi
\ifx\empty\@empty%
\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
\if\language\bbl@language\bbl@tempa%
\bbl@csarg\protected@edef{patterns@\bbl@tempa}{%\empty
\@ifundefined{bbl@patterns@\bbl@tempa}{}{%
\csname bbl@patterns@\bbl@tempa\endcsname\space#2}}%
\fi}
\end{verbatim}

9.5 Southeast Asian scripts

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

function Babel.linebreaking.add_before(func, pos)
  tex.print(\[
\noexpand\csname bbl@luahyphenate\endcsname\]
)
  if pos == nil then
    table.insert(Babel.linebreaking.before, func)
  else
    table.insert(Babel.linebreaking.before, pos, func)
  end
end

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

\def\bbl@intraspace#1 #2 #3\@@{\relax
  \directlua{
Babel = Babel or {}
Babel.intraspaces = Babel.intraspaces or {}
Babel.intraspaces[\csname bbl@sbcp@\languagename\endcsname] = {
b = #1, p = #2, m = #3
}
Babel.locale_props[\the\localeid].intraspace = {
b = #1, p = #2, m = #3
}
}
\def\bbl@intrapenalty#1\@@{\relax
  \directlua{
Babel = Babel or {}
Babel.intrapenalties = Babel.intrapenalties or {}
Babel.intrapenalties[\csname bbl@sbcp@\languagename\endcsname] = #1
Babel.locale_props[\the\localeid].intrapenalty = #1
}
\begingroup
\catcode\%=12\relax
\catcode\^=14\relax
\catcode\'=12\relax
\catcode\~=12\relax
\gdef\bbl@seaintraspace{^\relax
\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
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
for item in node.traverse(head) do
  if item.id == node.id'glyph' then
    last_char = item
  elseif i == 7 and item.subtype == 3 and last_char

111
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, Cyrll
    local intraspace = Babel.intraspaces[lg]
    local intrapenalty = Babel.intrapenalties[lg]
    local n
    if intrapenalty ~= 0 then
      n = node.new(14, 0)  -- penalty
      n.penalty = intrapenalty
      node.insert_before(head, item, n)
    end
    n = node.new(12, 13)  -- (glue, spaceskip)
    node.setglue(n, intraspace.b * quad,
                 intraspace.p * quad,
                 intraspace.m * quad)
    node.insert_before(head, item, n)
    node.remove(head, item)
  end
end
end
end
}

\bbl@luahyphenate

\catcode`%=14
\gdef\bbl@cjkintrinspace{%
\let\bbl@cjkintrospace=\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
    end
  end
end
}

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.

\catcode`%=14
\gdef\bbl@cjkintrospace{%
\let\bbl@cjkintrospace=\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
    end
  end
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 ~= \the\l@nohyphenation then
    local intrapenalty = props.intrapenalty
    if intrapenalty ~= 0 then
        local n = node.new(14, 0) % penalty
        n.penalty = intrapenalty
        node.insert_before(head, item, n)
    end
    local intraspace = props.intraspace
    local n = node.new(12, 13) % (glue, spaceskip)
    node.setglue(n, intraspace.b * quad,
                 intraspace.p * quad,
                 intraspace.m * quad)
    node.insert_before(head, item, n)
end
if font.getfont(item.font) then
    quad = font.getfont(item.font).size
end
last_class = class
last_lang = lang
else % if penalty, glue or anything else
    last_class = nil
end
lang.hyphenate(head)
}
\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')

113
\def\bbl@provide@intraspace{%  \bbl@ifunset{bbl@intsp@\languagename}{}%  \expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else  \bbl@xin{/c}{/\bbl@cl{lnbrk}}%  \ifin@ % cjk  \bbl@cjkintraspace  \directlua{  Babel = Babel or {}  Babel.locale_props = Babel.locale_props or {}  Babel.locale_props[\the\localeid].linebreak = 'c'  }%  \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\@@}%  \ifx\bbl@KVP@intrapenalty\@nnil  \bbl@intrapenalty0\@@%  \fi%  \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}}')  }%  \bbl@exp{\bbl@KVP@intrapenalty\@nil\bbl@intrapenalty0\@%  \fi%  \fi  \bbl@exp{\bbl@KVP@intrapenalty\@nil\fi}%  \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@%}\endgroup

9.7 Arabic justification

\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,0641,0642,0643,0644,0645,0646,0647,0649}%\begingroup\catcode`_=11 \catcode`:=11\gdef\bblar@nofswarn{\gdef\msg_warning:nnx##1##2##3{}%\endgroup\gdef\bbl@arabicjust{%  \let\bbl@arabicjust\relax  \newattribute\bblar@kashida\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')  }%  \directlua{  Babel.arabic.justify = Babel.arabic.justify or {}  Babel.arabic.justify = \catcode`\n=11\gdef\msg_warning:nnx##1##2##3{}\endgroup\bbl@arabicjust\relax%  \newattribute\bblar@kashida\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')  }%  \directlua{  Babel.arabic.justify = Babel.arabic.justify or {}  Babel.arabic.justify = \catcode`\n=11\gdef\msg_warning:nnx##1##2##3{}\endgroup\bbl@arabicjust\relax%
114
Save both node lists to make replacement. TODO. Save also widths to
make computations.
\def\bblar@fetchjalt#1#2#3#4{\bbl@exp{\\bbl@foreach{#1}}\bbl@ifunset{bblar@JE@##1}%
\setbox\z@\hbox{\char"##1#2}\setbox\z@\hbox{\char\@nameuse{bblar@JE@##1#2}}%
directlua{local last = nil
for item in node.traverse(tex.box[0].head) do
  if item.id == node.id'glyph' and item.char > 0x600 and
    not (item.char == 0x200D) then
    last = item
  end
end
Babel.arabic.#3['##1#4'] = last.char
}%

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?
\gdef\bbl@parsejalt{\ifx\addfontfeature\@undefined\else
\bbl@xin{/e}{/\bbl@cl{lnbrk}}\ifin@
\directlua{if Babel.arabic.elong_map[the\localeid]\fontid\font] == nil then
  Babel.arabic.elong_map[the\localeid]\fontid\font] = {}
  tex.print([\string\csname bbl@parsejalti\endcsname])
end\fi\fi}
\gdef\bbl@parsejalti{\begingroup\let\bbl@parsejalt\relax % To avoid infinite loop
\edef\bbl@tempb{\fontid\font} %
\directlua{if Babel.arabic.elong_map[the\localeid]\fontid\font] == nil then
  Babel.arabic.elong_map[the\localeid]\fontid\font] = {}
  tex.print([\string\csname bbl@parsejalti\endcsname])
end\fi}\endgroup}
\let\bbl@parsejalt\relax % To avoid infinite loop
\edef\bbl@tempb{\fontid\font}
\directlua{\bblar@fetchjalt\bblar@chars{\char664a}{from}{a} % Alef maksura
\bblar@fetchjalt\bblar@chars{\char6649}{from}{y} % Yeh
\addfontfeature{RawFeature=+jalt}%
\@namedef{bblar@JE@6643}{06AA}% todo: catch medial kaf
\bblar@fetchjalt\bblar@chars{\char6649}{dest}{y}
\directlua{for k, v in pairs(Babel.arabic.from) do
  if Babel.arabic.dest[k] and
    not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
    Babel.arabic.elong_map[the\localeid][\bbl@tempb]
  [Babel.arabic.from[k]] = Babel.arabic.dest[k]
end
\bblar@fetchjalt\bblar@chars{\char664a}{dest}{a}
\bblar@fetchjalt\bblar@chars{\char6649}{dest}{y} %
\directlua{for k, v in pairs(Babel.arabic.from) do
  if Babel.arabic.dest[k] and
    not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
    Babel.arabic.elong_map[the\localeid][\bbl@tempb]
  [Babel.arabic.from[k]] = Babel.arabic.dest[k]
end
\bblar@fetchjalt\bblar@chars{\char6649}{dest}{y} %
\directlua{
Babel.arabic = Babel.arabic or {}
Babel.arabic.from = {}
Babel.arabic.dest = {}
Babel.arabic.justify_factor = 0.95
Babel.arabic.justify_enabled = true
Babel.arabic.kashida_limit = -1

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

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

function Babel.arabic.justify_hlist(head, line, gc, size, pack)
local d, new
local k_list, k_item, pos_inline
local width, width_new, full, k_curr, wt_pos, goal, shift
local subst_done = false
local elong_map = Babel.arabic.elong_map
local cnt
local last_line
local GLYPH = node.id'glyph'
local KASHIDA = Babel.attr_kashida
local LOCALE = Babel.attr_locale
if line == nil then
  line = {}
  line.glue_sign = 1
  line.glue_order = 0
  line.head = head
  line.shift = 0
  line.width = size
end

% 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
  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} )
note.set_attribute(n.prev, KASHIDA, 0)
end
end

% Tatwil
if Babel.kashida_wts then
local k_wt = node.get_attribute(n, KASHIDA)
if k_wt > 0 then % todo. parameter for multi inserts
table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
end
end
end

end % of node.traverse_id
if #elongs == 0 and #k_list == 0 then goto next_line end
full = line.width
shift = line.shift
goal = full * Babel.arabic.justify_factor % A bit crude
width = node.dimensions(line.head) % The 'natural' width

% == Elongated ==
% Original idea taken from 'chikenize'
while (#elongs > 0 and width < goal) do
subst_done = true
local x = #elongs
local curr = elongs[x].node
local oldchar = curr.char
curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
width = node.dimensions(line.head) % Check if the line is too wide
% Substitute back if the line would be too wide and break:
if width > goal then
curr.char = oldchar
break
end
% If continue, pop the just substituted node from the list:
table.remove(elongs, x)
end

% == Tatwil ==
if #k_list == 0 then goto next_line end
width = node.dimensions(line.head) % The 'natural' width
k_curr = #k_list % Traverse backwards, from the end
wt_pos = 1

while width < goal do
subst_done = true
k_item = k_list[k_curr].node
if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
d = node.copy(k_item)
d.char = 0x0640
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
width = width_new
end
if k_curr == 1 then
k_curr = #k_list

end % of node.traverse_id
wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1

else
    k_curr = k_curr - 1
end

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

9.8 Common stuff

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

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 short function which just traverse the node list to carry out the replacements. The table 
loc_to_scr gets the locale form a script range (note the locale is the key, and that there is an 
intermediate table built on the fly for optimization). This locale is then used to get the \language 
and the localeid 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 discretionarys 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}},
    ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},}
'Copt' = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
['Cyrl'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
{0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
{0xAB00, 0xAB2F}},
['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
['Hans'] = {{0x02EB0, 0x2E7F}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
{0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
{0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
{0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
{0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
{0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
['Hebr'] = {{0x0590, 0x05FF}},
['Hant'] = {{0x0300, 0x03FF}, {0x3040, 0x309F}, {0x30A0, 0x30FF}},
['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
'Knda' = {{0x0C80, 0x0CFF}},
['Kore'] = {{0x0A00, 0x0AFF}, {0x1100, 0x11FF}, {0x3130, 0x318F},
{0x4E00, 0x9FFF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
{0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
['Laoo'] = {{0x0E80, 0x0EFF}},
['Latn'] = {{0x0000, 0x007F}, {0x0100, 0x024F}, {0x1E00, 0x1EFF},
{0x2C60, 0x2C7F}, {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
['Mlym'] = {{0x0D00, 0x0D7F}},
['Mymr'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0180, 0x024F},
{0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
{0x4E00, 0x9FF}, {0x9A90, 0x9AFF}, {0xA960, 0xA97F},
{0x4E00, 0x9FF}, {0xF900, 0xFAFF}, {0xF900, 0xFAFF},
{0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
{0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
{0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
['Mlym'] = {{0x0D00, 0x0D7F}},
['Orya'] = {{0x0B00, 0x0BFF}},
['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
['Thai'] = {{0x0E00, 0x0EFF}},
['Tibt'] = {{0x0F00, 0x0FFF}},
['Telu'] = {{0x0C00, 0x0C7F}},
['Tfng'] = {{0x2D30, 0x2D7F}},
['Telu'] = {{0x0C00, 0x0C7F}},
['Tfng'] = {{0x2D30, 0x2D7F}},
['Thai'] = {{0x0E00, 0x0EFF}},
['Vaii'] = {{0xA500, 0xA63F}},
['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
}
if item.char >= rg[1] and item.char <= rg[2] then
    Babel.chr_to_loc[item.char] = lc
toloc = lc
    break
end
end
end
end
% Now, take action, but treat composite chars in a different
% fashion, because they 'inherit' the previous locale. Not yet
% optimized.
if not toloc and
    (item.char >= 0x0300 and item.char <= 0x036F) or
    (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
    (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
    toloc = toloc_save
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 toloc > -1 then
    if Babel.locale_props[toloc].lg then
        item.lang = Babel.locale_props[toloc].lg
        node.set_attribute(item, LOCALE, toloc)
    end
    if Babel.locale_props[toloc]['/..item.font'] then
        item.font = Babel.locale_props[toloc]['/..item.font']
    end
    toloc_save = toloc
end
elseif not inmath and item.id == 7 then % Apply recursively
    item.replace = item.replace and Babel.locale_map(item.replace)
    item.pre = item.pre and Babel.locale_map(item.pre)
    item.post = item.post and Babel.locale_map(item.post)
elseif item.id == node.id'math' then
    inmath = (item.subtype == 0)
end
return head
end
}

The code for \babelcharproperty is straightforward. Just note the modified lua table can be
different.
\newcommand\babelcharproperty[1]{%
\count@=#1\relax
\ifvmode
\expandafter\bbl@chprop
\else
\bbl@error{\string\babelcharproperty\space can be used only in vertical mode (preamble or between paragraphs)}%\See the manual for futher info}%
\fi}
\newcommand\bbl@chprop[3][\the\count@]{%
\@tempcnta=#1\relax
\@tempcntb=#2\relax
\ifnum\@tempcnta<1
\bbl@ifunset{bbl@chprop@#3}%
{\bbl@error{No property named '#3'. Allowed values are\ direction (bc), mirror (bmg), and linebreak (lb)}%\See the manual for futher info}%
\else
\bbl@chprop[\@tempcntb][\the\count@]{}
\fi}%
\loop
120
Post-handling hyphenation patterns for non-standard rules, like ff to ff-f. There are still some issues with speed (not very slow, but still slow). The Lua code is below.

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

\begin{verbatim}
\def\bbl@chprop@directionI{% \\
\directlua{ \texttt{Babel.characters[the\count@] = Babel.characters[the\count@] or \{} \\
Babel.characters[the\count@][\texttt{'d'}] = \texttt{'#1'} \\
\} } \\
\let\bbl@chprop@bc\bbl@chprop@direction \\
\def\bbl@chprop@mirrorI{% \\
\directlua{ \texttt{Babel.characters[the\count@] = Babel.characters[the\count@] or \{} \\
Babel.characters[the\count@][\texttt{'m'}] = \texttt{\number#1'} \\
\} } \\
\let\bbl@chprop@bmg\bbl@chprop@mirror \\
\def\bbl@chprop@linebreakI{% \\
\directlua{ \texttt{Babel.cjk_characters[the\count@] = Babel.cjk_characters[the\count@] or \{} \\
Babel.cjk_characters[the\count@][\texttt{'c'}] = \texttt{\texttt{\#1}}' \\
\} } \\
\let\bbl@chprop@lb\bbl@chprop@linebreak \\
\def\bbl@chprop@localeI{% \\
\directlua{ \texttt{Babel.chr_to_loc = Babel.chr_to_loc or \{} \\
Babel.chr_to_loc[the\count@] = \texttt{\texttt{\bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space'}} \\
\} } \\
\end{verbatim}

\begin{verbatim}
\begingroup \\
\catcode`\-=12 \\
\catcode`\%=12 \\
\catcode`\&=14 \\
\catcode`\|=12 \\
\gdef\babelprehyphenation{% \\
@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}] \\
\gdef\babelposthyphenation{% \\
@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}] \\
\gdef\bbl@settransform#1[#2]#3#4#5{% \\
\ifcase#1 \\
\bbl@activateprehyphen \\
\or \\
\bbl@activateposthyphen \\
\fi \\
\begingroup \\
\def\babeltempa{\bbl@add@list\babeltempb}&% \\
\let\babeltempb\@empty \\
\def\bbl@tempa{#5}&% \\
\bbl@replace\bbl@tempa{,}{ ,}&% TODO. Ugly trick to preserve {} \\
\endgroup \\
\endverbatim

\end{verbatim}
local rep = ['remove']
rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
rep = rep:gsub('^%s*(insert)%s*$', 'insert = true, ') 
rep = rep:gsub('(string)%s*=%s*(["%s,]*)', Babel.capture_func)
if #1 == 0 or #1 == 2 then
  rep = rep:gsub('(space)%s*=%s*([^%s,\]+)%s*([^%s,\]+)%s*([^%s,\]+)',
    'space = {' .. '%2, %3, %4' .. '}
  rep = rep:gsub('(spacefactor)%s*=%s*([^%s,\]+)%s*([^%s,\]+)%s*([^%s,\]+)',
    'spacefactor = {' .. '%2, %3, %4' .. '}
  rep = rep:gsub('(kashida)%s*=%s*([^%s,]*)', Babel.capture_kashida)
else
  rep = rep:gsub( '(no)%s*=%s*([^%s,]*)', Babel.capture_func)
  rep = rep:gsub( '(pre)%s*=%s*([^%s,]*)', Babel.capture_func)
  rep = rep:gsub( '(post)%s*=%s*([^%s,]*)', Babel.capture_func)
end
tex.print(['{' .. rep .. '}'])
}
local lbkr = Babel.linebreaking.replacements[#1]
local u = unicode.utf8
local id, attr, label
if #1 == 0 then
  id = \the\csname bbl@id@@#3\endcsname\space
else
  id = \the\csname l@#3\endcsname\space
end
\ifx\bbl@kv@attribute\relax
  attr = -1
\else
  attr = \luatexbase.registernumber'\bbl@kv@attribute'
\fi
\ifx\bbl@kv@label\relax
  label = \s==[\bbl@kv@label]==
\fi
&% Convert pattern:
local patt = \string.gsub([==[#4]==], '%s', '')
if #1 == 0 then
  patt = \string.gsub(patt, '|', ' ')
end
if not u.find(patt, '()', nil, true) then
  patt = '()' .. patt .. '()'
end
if #1 == 1 then
  patt = \string.gsub(patt, '%(%)%^', '^()')
  patt = \string.gsub(patt, '%$%(%)', '()$')
end
patt = u.gsub(patt, '{(.)}',
  function (n)
    return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
  end)
patt = u.gsub(patt, '{(%x%x%x%x+)}',
  function (n)
    return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
  end)
lbkr[id] = lbkr[id] or {}
table.insert(lbkr[id],
  { label=label, attr=attr, pattern=patt, replace={\babeltempb} })\)&%
\endgroup
\let\bbl@transfont@list@empty
\def\bbl@settransfont{%
\global\let\bbl@settransfont\relax % Execute only once
\def\bbl@transfont{%
  \def\bbl@elt####1####2####3{%
  \bbl@ifblank{####3}{% Do nothing if no fonts
  \bbl@fforeach{####3}{%\def\bbl@tempe{########1}%
    \edef\bbl@tempd{########1}%
    \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
    \ifx\bbl@tempd\bbl@tempe
      \count@\@ne
    \else
      \count@\@ne
    \fi
  \if\count@\@ne%\fi
  \bbl@ifblank{####2}{%\def\bbl@temppe{########1}%
   \edef\bbl@temppe{\bbl@transfam/\f@series/\f@shape}%
   \ifx\bbl@temppe\bbl@tempe
     \count@\@ne
   \else
     \count@\@ne
   \fi
   \if\count@\@ne%\fi
  \bbl@ifblank{####1}{%\def\bbl@temppe{########1}%
    \edef\bbl@temppe{\bbl@transfam/\f@series/\f@shape}%
    \ifx\bbl@temppe\bbl@tempe
      \count@\@ne
    \else
      \count@\@ne
    \fi
    \if\count@\@ne%\fi
\bbl@csarg\unsetattribute{ATR@####2@####1@####3}
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 luaofload is applied, which is loaded by default by \LaTeX. Just in case, consider the possibility it has not been loaded.

\def\bbl@activate@preotf{% 
\let\bbl@activate@preotf\relax % only once 
\directlua{ 
Babel = Babel or {} 
% function Babel.pre_otfload_v(head) 
if Babel.numbers and Babel.digits_mapped then 
  head = Babel.numbers(head) 
end 
if Babel.bidi_enabled then 
  head = Babel.bidi(head, false, dir) 
end 
} 
}
function Babel.pre_otfload_h(head, gc, sz, pt, dir)
  if Babel.numbers and Babel.digits_mapped then
    head = Babel.numbers(head)
  end
  if Babel.bidi_enabled then
    head = Babel.bidi(head, false, dir)
  end
  return head
end

luatexbase.add_to_callback('pre_linebreak_filter',
  Babel.pre_otfload_v,
  'Babel.pre_otfload_v',
  luatexbase.priority_in_callback('pre_linebreak_filter',
    'luaotfload.node_processor') or nil)

luatexbase.add_to_callback('hpack_filter',
  Babel.pre_otfload_h,
  'Babel.pre_otfload_h',
  luatexbase.priority_in_callback('hpack_filter',
    'luaotfload.node_processor') or nil)
}

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

\breakafterdirmode=1
\ifnum\bbl@bidimode=\me % Any bidi= except default=1
  \let\bbl@beforeforeign\leavevmode
\AtEndOfPackage{\EnableBabelHook{babel-bidi}}
\RequirePackage{luatexbase}
\bbl@activate@preotf
\directlua{
  require('babel-data-bidi.lua')
  \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
    require('babel-bidi-basic.lua')
  \else
    require('babel-bidi-basic-r.lua')
  \fi}
\newattribute\bbl@attr@dir
\directlua{\bbl@attr@dir = \exp\exp\exp\output{\bodydir\pagedir\the\output}}
\bbl@exp{\output{\bodydir\pagedir\the\output}}
\chardef\bbl@thetextdir\z@
\chardef\bbl@thepardir\z@
\def\bbl@getluadir#1{%
  \directlua{
    if tex.#1dir == 'TLT' then
      tex.sprint('0')
    elseif tex.#1dir == 'TRT' then
      tex.sprint('1')
    end}
  \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
    \ifcase#3\relax
      \ifcase\bbl@getluadir{#1}\relax
        #2 TLT\relax
      \else
        #2 TRT\relax
    \fi
  \else
    \ifcase\bbl@getluadir{#1}\relax
      #2 TLT\relax
    \fi
  \endcase
}\else
RTL text inside math needs special attention. It affects not only to actual math stuff, but also to 'tabular', which is based on a fake math.

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 relevant.

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

\@hangfrom is useful in many contexts and it is redefined always with the layout option. There are, however, a number of issues when the text direction is not the same as the box direction (as set by \bodydir), and when \parbox and \hangindent are involved. Fortunately, latest releases
of luatex simplify a lot the solution with \shapemode.
With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a
modification could be applied to several classes and packages. Now, tabular seems to work (at least
in simple cases) with array, tabularx, hline, colortbl, longtable, booktabs, etc. However, dcolumn still
fails.

\begin{verbatim}
\bbl@trace{Redefinitions for bidi layout}
\%\langle More package options \rangle ≡
\chardef\bbl@eqnpos\z@\% Any bidi=
\ifx\matheqdirmode\@undefined\
\matheqdirmode@ne\% A luatex primitive
\fi
\def\bbl@eqdel{}\% \normalfont\normalcolor
\expandafter\@firstoftwo\bbl@eqdel\theequation\expandafter\@secondoftwo\bbl@eqdel
\def\bbl@eqdel#1{\hbox{#1} \hb@xt@.01pt{\hss\hb@xt@\displaywidth{\hss#1}\hss}}
\def\bbl@eqno@flip#1{\ifdim\predisplaysize=-\maxdimen\eqno
\hb@xt@.01pt{\hb@xt@\displaywidth{\hss#1}\hss} \else\leqno\hbox{#1}\fi}
\def\bbl@leqno@flip#1{\ifdim\predisplaysize=-\maxdimen\leqno
\hb@xt@.01pt{\hss\hb@xt@\displaywidth{\hss#1}\hss} \else\eqno\hbox{#1}\fi}
\AtBeginDocument{\
\if\bbl@noamsmath\relax\else
\ifx\maketag@@@\@undefined% Normal equation, eqnarray
\AddToHook{env/equation/begin}{\ifnum\bbl@thetextdir>\z@
\def\bbl@mathboxdir{\def\bbl@insidemath{1}}\let\@eqnnum\bbl@eqnum\edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}\chardef\bbl@thetextdir\z@
\bbl@add{\normalfont{\bbl@eqnodir}}\ifcase\bbl@eqnpos\let\bbl@puteqno\bbl@eqno@flip\or\let\bbl@puteqno\bbl@leqno@flip\fi\if\eqn\hb@xt@.01pt{\hb@xt@\displaywidth{\hss\hb@xt@\displaywidth{\hss#1}\hss} \else\eqno\hbox{#1}\fi\fi}
\AddToHook{env/eqnarray/begin}{\if\bbl@thetextdir>\z@
\def\bbl@mathboxdir{\def\bbl@insidemath{1}}\let\@eqnnum\bbl@eqnum\edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}\chardef\bbl@thetextdir\z@
\bbl@add{\normalfont{\bbl@eqnodir}}\ifcase\bbl@eqnpos\let\bbl@puteqno\bbl@eqno@flip\or\let\bbl@puteqno\bbl@leqno@flip\fi\or\let\bbl@puteqno\bbl@leqno@flip\fi
\fi}
\if\bbl@noamsmath\relax\else
\ifx\maketag@@@\@undefined\% Normal equation, eqnarray
\AddToHook{env/equation/begin}{\ifnum\bbl@thetextdir>\z@
\def\bbl@mathboxdir{\def\bbl@insidemath{1}}\let\@eqnnum\bbl@eqnum\edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}\chardef\bbl@thetextdir\z@
\bbl@add{\normalfont{\bbl@eqnodir}}\ifcase\bbl@eqnpos\let\bbl@puteqno\bbl@eqno@flip\or\let\bbl@puteqno\bbl@leqno@flip\fi\or\let\bbl@puteqno\bbl@leqno@flip\fi
\AddToHook{env/eqnarray/begin}{\if\bbl@thetextdir>\z@
\def\bbl@mathboxdir{\def\bbl@insidemath{1}}\let\@eqnnum\bbl@eqnum\edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}\chardef\bbl@thetextdir\z@
\bbl@add{\normalfont{\bbl@eqnodir}}\ifcase\bbl@eqnpos\let\bbl@puteqno\bbl@eqno@flip\or\let\bbl@puteqno\bbl@leqno@flip\fi\or\let\bbl@puteqno\bbl@leqno@flip\fi
\fi}
\end{verbatim}

127
Very likely the \textoutput routine must be patched in a quite general way to make sure the \pagedir is set to \pagedir. Note outside \textoutput they can be different (and often are). For the moment, two \textit{ad hoc} changes.

\AtBeginDocument{%
  @ifpackageloaded{colortbl}%
  {\if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\tabclassz
    {<\ifcase><@chnum>}}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
  \if\@tabclassz% Hide conditionals
    \mbox\@sreplace\\@tabclassz
    {<\ifcase><@chnum>}%
OMEGA provided a companion to \mathdir (\nextfakemath) for those cases where we did not want it to be applied, so that the writing direction of the main text was left unchanged. \bbl@nextfake is an attempt to emulate it, because luatex has removed it without an alternative. Also, \hangindent does not honour direction changes by default, so we need to redefine \@hangfrom.

\ifnum\bbl@bidimode>\z@ % Any bidi=
\def\bbl@nextfake#1{% non-local changes, use always inside a group!
  \def\bbl@insidemath{0}\
  \mathdir\the\bodydir
  #1% Once entered in math, set boxes to restore values
  \ifmmode\everyvbox{\the\everyvbox}\
  \everyhbox{\the\everyhbox}\
  \everyvbox{\the\everyvbox}}\fi
\def\@hangfrom#1{\setbox\@tempboxa\hbox{{#1}}\hangindent\wd\@tempboxa\ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else\shapemode\@ne\fi
\noindent\box\@tempboxa}
\fi
\IfBabelLayout{tabular}{{\let\bbl@OL@@tabular\@tabular\bbl@replace\@tabular{$}{\bbl@nextfake$}\let\bbl@NL@@tabular\@tabular\AtBeginDocument{%\if\bbl@NL@@tabular\@tabular\else\bbl@replace\@tabular{$}{\bbl@nextfake$}%\let\bbl@NL@@tabular\@tabular\fi}}
\fi
\IfBabelLayout{lists}{{\let\bbl@OL@list\list\bbl@sreplace\list{\parshape}{\bbl@listparshape}%\ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else\shapemode\tw@\fi}}
\fi
\IfBabelLayout{graphics}{{\let\bbl@pictresetdir\relax\def\bbl@pictsetdir#1{\ifcase\bbl@thetextdir\let\bbl@pictresetdir\relax\else\parshape #1 #2 #3 %\ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else\shapemode\tw@\fi}}}
\fi
\IfBabelLayout{}{{}}
\ifeqcase#1\bodydir TLT % Remember this sets the inner boxes
\or\textdir TLT
\else\bodydir TLT \textdir TLT\fi
% \text|par)dir required in pgf:
\def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
\if\textdir TLT\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
\latexbase.add_to_callback("hpack_filter", Babel.picture_dir,
"Babel.picture_dir")
}%
\AtBeginDocument{%
\def\LS@rot{%
\setbox\@outputbox\vbox{%
\hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}}%
\long\def\put(#1,#2)#3{%
%@killglue
% Try:
\ifx\bbl@pictresetdir\relax
\def\bbl@tempc{0}%
\else
\directlua{
Babel.get_picture_dir = true
Babel.picture_has_bidi = 0
}%
\setbox\z@\hb@xt@\z@{%
%@defaultunitsset\@tempdimc{#1}\unitlength
 \kern\@tempdimc
 #3\hss}% TODO: #3 executed twice (below). That's bad.
\edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
\fi
% Do:
\%@defaultunitsset\@tempdimc{#2}\unitlength
\raise\@tempdimc{hbox@xt@\z@{%
%@defaultunitsset\@tempdimc{#1}\unitlength
 \kern\@tempdimc
 %3\hss}%
\edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
\ignorespaces}%
\MakeRobust\put}%
\AtBeginDocument{%
\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir@gobble}%
\if\pgfpicture\undefined\else % TODO. Allow deactivate?
\AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\one}%
\bbl@addpgfinterruptpicture{\bbl@pictsetdir%}
\bbl@addpgfsys@beginpicture{\bbl@pictsetdir\z@}%
\fi
\if\tikzpicture\undefined\else
\AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\tw@}%
\bbl@addtikz@atbegin@node{\bbl@pictsetdir}%
\bbl@sreplace\tikz{"\beginingroup\bbl@pictsetdir\tw@}%
\fi
\if\tcolorbox\undefined\else
\fi
\fi

132
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.
9.12 Lua: transforms

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

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

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

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

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

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

-- Here and above we remove some trailing chars but not the
-- corresponding nodes. But they aren't accessed.
if word_string:sub(-1) == ' ' then
    word_string = word_string:sub(1,-2)
end
word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
return word_string, word_nodes, item, lang
end
Babel.fetch_subtext[1] = function(head)
    local word_string = ''
    local word_nodes = {}
    local lang
    local item = head
    local inmath = false

    while item do
        if item.id == 11 then
            inmath = (item.subtype == 0)
        end
        if inmath then
            -- pass
        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
                end
                word_string = word_string .. unicode.utf8.char(item.char)
                word_nodes[#word_nodes+1] = item
            end
            else
                break

        end
end
elseif item.id == 7 and item.subtype == 2 then
    word_string = word_string .. '='
    word_nodes[#word_nodes+1] = item
elseif item.id == 7 and item.subtype == 3 then
    word_string = word_string .. '|
    word_nodes[#word_nodes+1] = item
-- (1) Go to next word if nothing was found, and (2) implicitly
-- remove leading USs.
elseif word_string == '' then
    -- pass
elseif (item.id == 12 and item.subtype == 13) then
    break
else
    word_string = word_string .. Babel.us_char
    word_nodes[#word_nodes+1] = item -- Will be ignored
end
item = item.next
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
    -- substrings.
    if string.find(w:sub(first, last-1), Babel.us_char) then break end
    local save_last = last -- with A()BC()D, points to D
    -- Fix offsets, from bytes to unicode. Explained above.
    first = u.len(w:sub(1, first-1)) + 1
    last = u.len(w:sub(1, last-1)) -- now last points to C
    -- This loop stores in a small table the nodes
    -- corresponding to the pattern. Used by 'data' to provide a
    -- predictable behavior with 'insert' (w_nodes is modified on
    -- the fly), and also access to 'remove'd nodes.
    local sc = first-1 -- Used below, too
    local data_nodes = {}
    local enabled = true
    for q = 1, last-first+1 do
      data_nodes[q] = w_nodes[sc+q]
      if enabled
        and attr > -1
        and not node.has_attribute(data_nodes[q], attr)
        then
          enabled = false
      end
    end
    -- This loop traverses the matched substring and takes the
    -- corresponding action stored in the replacement list.
    -- sc = the position in substr nodes / string
    -- rc = the replacement table index
local rc = 0

while rc < last-first+1 do -- for each replacement
    if Babel.debug then
        print('.....', rc + 1)
    end
    sc = sc + 1
    rc = rc + 1

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

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

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

if crep then
    step = crep.step or 0
end

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

-- 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)
ret = ret:gsub('%\[%\[%\]%\]%.%.', '')
ret = ret:gsub%.%\[%\[%\]%\]''
return key .. [[=function(m) return ]] .. ret .. [[ end]]
end

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

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

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) break end end elseif table.getn(Babel.kashida_wts) == p then table.insert(Babel.kashida_wts, wt) break end
end
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)
  end
  n.char = s
  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)

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'},
[0x28]={d='on', m=0x29},
[0x29]={d='on', m=0x28},
[0x2A]={d='on'},
[0x2B]={d='es'},
[0x2C]={d='cs'},
```

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

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

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

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

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

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

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

   local dir, dir_real
   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 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
               end
            end
         end
      end
   end
```

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

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

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

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

By W2, there are no <en> <et> <es> if strong == <al>, only <an>. Therefore, there are not <et en>
nor <en et>, W5 can be ignored, and W6 applied:

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

Numbers in R mode. A sequence of <en>, <et>, <an>, <es> and <cs> is typeset (with some rules) in L
mode. We store the starting and ending points, and only when anything different is found (including
nil, ie, a non-char), the textdir is set. This means you cannot insert, say, a 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>.

if dir == 'en' or dir == 'an' or dir == 'et' then
if dir ~= 'et' then

type_n = dir

end

first_n = first_n or item

last_n = last_es or item

last_es = nil

elseif dir == 'es' and last_n then -- W3+W6

last_es = item

elseif dir == 'cs' then -- it's right - do nothing

elseif first_n then -- & if dir = any but en, et, es, cs, inc nil

if strong_lr == 'r' and type_n ~= '' then

dir_mark(head, first_n, last_n, 'r')

elseif strong_lr == 'l' and first_d and type_n == 'an' then

dir_mark(head, first_n, last_n, 'r')

dir_mark(head, first_d, last_d, outer)

first_d, last_d = nil, nil

elseif strong_lr == 'l' and type_n ~= '' then

last_d = last_n

end

type_n = ''

first_n, last_n = nil, nil

end

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

if dir == 'l' or dir == 'r' then

if dir ~= outer then

first_d = first_d or item

last_d = item

elseif first_d and dir ~= strong_lr then

dir_mark(head, first_d, last_d, outer)

first_d, last_d = nil, nil

end

end

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

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

if dir and not last_lr and dir ~= 'l' and outer == 'r' then

item.char = characters[item.char] and

characters[item.char].m or item.char

elseif (dir or new_dir) and last_lr ~= item then

local mir = outer .. strong_lr .. (dir or outer)

if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then

for ch in node.traverse(node.next(last_lr)) do

if ch == item then break end

if ch.id == node.id'Glyph' and characters[ch.char] then

ch.char = characters[ch.char].m or ch.char

end

end

end

end

Save some values for the next iteration. If the current node is ‘dir’, open a new sequence. Since dir
could be changed, strong is set with its real value (dir_real).

if dir == 'l' or dir == 'r' then

last_lr = item

strong = dir_real -- Don't search back - best save now

strong_lr = (strong == 'l') and 'l' or 'r'

elseif new_dir then

last_lr = nil

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

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

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

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

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

return node.prev(head) or head

end

⟨/basic-r⟩

And here the Lua code for bidi=basic:

Babel = Babel or {}

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

Babel.fontmap = Babel.fontmap or {}

Babel.fontmap[0] = {} -- l
Babel.fontmap[1] = {} -- r

Babel.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)
    new_state.sim, new_state.eim = nil, nil
  end

  return head, new_state
end

local function insert_numeric(head, state)

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

return new_state.sim, new_state.eim = nil, nil

return head, new_state
end

local function insert_numeric(head, state)

local new
local new_state = state

return new

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

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

function Babel.bidi(head, ispar, hdir)
  local d -- d is used mainly for computations in a loop
  local prev_d = ''
  local new_d = false
  local nodes = {}
  local outer_first = nil
  local inmath = false
  local glue_d = nil
  local glue_i = nil
  local has_en = false
  local first_et = nil
  local has_hyperlink = false
  local ATDIR = Babel.attr_dir
  local save_outer
  local temp = node.get_attribute(head, ATDIR)
  if temp then
    temp = temp & 0x3
    save_outer = (temp == 0 and 'l') or
      (temp == 1 and 'r') or
      (temp == 2 and 'al')
  elseif ispar then -- Or error? Shouldn't happen
    save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
  else -- Or error? Shouldn't happen
    save_outer = ('TRT' == hdir) and 'r' or 'l'
  end
  -- when the callback is called, we are just _after_ the box,
  -- and the textdir is that of the surrounding text
  -- if not ispar and hdir ~= tex.textdir then
    -- save_outer = ('TRT' == hdir) and 'r' or 'l'
  -- end
  local outer = save_outer
  local last = outer
  if save_outer == 'al' then save_outer = 'r' end
  local fontmap = Babel.fontmap

for item in node.traverse(head) do

-- In what follows, #node is the last (previous) node, because the
-- current one is not added until we start processing the neutrals.

-- three cases: glyph, dir, otherwise
if item.id == GLYPH
    or (item.id == 7 and item.subtype == 2) then

    local d_font = nil
    local item_r
    if item.id == 7 and item.subtype == 2 then
        item_r = item.replace -- automatic discs have just 1 glyph
    else
        item_r = item
    end

    local chardata = characters[item_r.char]
    d = chardata and chardata.d or nil
    if not d or d == 'nsm' then
        for nn, et in ipairs(ranges) do
            if item_r.char < et[1] then
                break
            elseif item_r.char <= et[2] then
                if not d then d = et[3]
                elseif d == 'nsm' then d_font = et[3]
                end
            end
            break
        end
    end
    d = d or 'l'

-- A short 'pause' in bidi for mapfont
    d_font = d_font or d
    d_font = (d_font == 'l' and 0) or
        (d_font == 'nsm' and 0) or
        (d_font == 'r' and 1) or
        (d_font == 'al' and 2) or
        (d_font == 'an' and 2) or nil
    if d_font and fontmap and fontmap[d_font][item_r.font] then
        item_r.font = fontmap[d_font][item_r.font]
    end

    if new_d then
        table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
    if inmath then
        attr_d = 0
    else
        attr_d = node.get_attribute(item, ATDIR)
        attr_d = attr_d & 0x3
    end
    if attr_d == 1 then
        outer_first = 'r'
        last = 'r'
    elseif attr_d == 2 then
        outer_first = 'r'
        last = 'al'
    else
        outer_first = 'l'
        last = 'l'
    end
    outer = last
    has_en = false
``` lua
7460  first_et = nil
7461  new_d = false
7462  end
7463
7464  if glue_d then
7465      if (d == 'l' and 'l' or 'r') ~= glue_d then
7466          table.insert(nodes, {glue_i, 'on', nil})
7467      end
7468      glue_d = nil
7469      glue_i = nil
7470  end
7471
7472  elseif item.id == DIR then
7473      d = nil
7474
7475      if head ~= item then new_d = true end
7476  elseif item.id == node.id'glue' and item.subtype == 13 then
7477      glue_d = d
7478      glue_i = item
7479      d = nil
7480  elseif item.id == node.id'math' then
7481      inmath = (item.subtype == 0)
7482  elseif item.id == 8 and item.subtype == 19 then
7483      has_hyperlink = true
7484  else
7485      d = nil
7486  end
7487
7488  -- AL <= EN/ET/ES -- W2 + W3 + W6
7489  if last == 'al' and d == 'en' then
7490      d = 'an' -- W3
7491  elseif last == 'al' and (d == 'et' or d == 'es') then
7492      d = 'on' -- W6
7493  end
7494
7495  -- EN + CS/ES + EN -- W4
7496  if d == 'en' and #nodes >= 2 then
7497      if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7498          and nodes[#nodes-1][2] == 'en' then
7499          nodes[#nodes][2] = 'en'
7500      end
7501  end
7502
7503  -- AN + CS + AN -- W4 too, because uax9 mixes both cases
7504  if d == 'an' and #nodes >= 2 then
7505      if (nodes[#nodes][2] == 'cs')
7506          and nodes[#nodes-1][2] == 'an' then
7507          nodes[#nodes][2] = 'an'
7508      end
7509  end
7510
7511  -- ET/EN -- W5 + W7->l / W6->on
7512  if d == 'et' then
7513      first_et = first_et or (#nodes + 1)
7514  elseif d == 'en' then
7515      has_en = true
7516  elseif first_et then -- d may be nil here!
7517      if has_en then
```

149
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
end
-- dummy node, to close things
table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})

-------------- NEUTRAL ---------------
local first_on = nil
for q = 1, #nodes do
  local item
  outer = nodes[q][3]
  last = outer_first or last
  local d = nodes[q][2]
  if d == 'an' or d == 'en' then d = 'r' end
  if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end -- W6
  if d == 'on' then
    first_on = first_on or q
  elseif first_on then
    if last == d then
      temp = d
    else
      temp = outer
    end
  end
  for r = first_on, q - 1 do
    nodes[r][2] = temp
    item = nodes[r][1] -- MIRRORING
    if Babel.mirroring_enabled and item.id == GLYPH
      and temp == 'r' and characters[item.char] then
      local font_mode = ''
      if item.font > 0 and font.fonts[item.font].properties then
        font_mode = font.fonts[item.font].properties.mode
      end
      if font_mode ~= 'harf' and font_mode ~= 'plug' then
        item.char = characters[item.char].m or item.char
      end
    end
  end
  first_on = nil
end
if d == 'r' or d == 'l' then last = d end
end

-------------- IMPLICIT, REORDER ----------------
outer = save_outer
last = outer
local state = {}
state.has_r = false
for q = 1, #nodes do
  local item = nodes[q][1]
  outer = nodes[q][3] or outer
  local d = nodes[q][2]
  if d == 'nsm' then d = last end -- W1
  if d == 'en' then d = 'an' end
  local isdir = (d == 'r' or d == 'l')
  if outer == 'l' and d == 'an' then
    state.san = state.san or item
    state.ean = item
  elseif state.san then
    state.san = state.san or item
    state.ean = item
  elseif state.san then

151
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 d == 'on' and state.san then
  state.san = state.san or item
  state.ean = item
end
head = node.prev(head) or head
-------------- FIX HYPERLINKS ----------------
if has_hyperlink then
  local flag, linking = 0, 0
  for item in node.traverse(head) do
    if item.id == DIR then
      if item.dir == '+TRT' or item.dir == '+TLT' then
        flag = flag + 1
      elseif item.dir == '-TRT' or item.dir == '-TLT' then
        flag = flag - 1
      end
    elseif item.id == 8 and item.subtype == 19 then
      linking = flag
    elseif item.id == 8 and item.subtype == 20 then
      if linking > 0 then
        if item.prev.id == DIR and
           (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
          d = node.new(DIR)
          d.dir = item.prev.dir
          node.remove(head, item.prev)
          node.insert_after(head, item, d)
        end
      end
      linking = 0
    end
  end
end
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:

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

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

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.

```
\ProvidesLanguage{nil}[⟨⟨date⟩⟩ v⟨⟨version⟩⟩] Nil language
\LdfInit{nil}{datenil}
```

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

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

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

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

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

```
\caption{\nil}
\datenil
```

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

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

\begin{_document}
\ldf@finish
\langle \langle \text{Compute Julian day} \rangle \rangle ≡ \\
\def\bbl@fpmod#1#2{(#1-#2*floor(#1/#2))} \\
\def\bbl@cs@gregleap#1{\textbf{}} \\
\def\bbl@cs@jd#1#2#3{ % year, month, day \\
\fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) + \\
floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) + \\
floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) + \\
((#2 <= 2) ? 0 : (\textbf{bbl@cs@gregleap}#1 ? -1 : -2)) + #3) }% \\
\langle \langle \text{Compute Julian day} \rangle \rangle
\end{document}

12.1 Islamic

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

\begin{document}
\ExplSyntaxOn
\langle \langle \text{Compute Julian day} \rangle \rangle
\ExplSyntaxOff
\% == islamic (default) 
\% Not yet implemented
\langle \langle \text{Compute Julian day} \rangle \rangle
\end{document}

The Civil calendar.

\begin{document}
\explSyntaxOn\langle \langle \text{Compute Julian day} \rangle \rangle
\ExplSyntaxOff
\% == islamic (default) 
\% Not yet implemented
\end{document}
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 $\sim 1435/\sim 1460$ (Gregorian $\sim 2014/\sim 2038$).

\begin{verbatim}
7827 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqra{x}}
\@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqra{+1}}
\@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqra{-1}}
\def\bbl@ca@islamcuqra{x}{\bbl@foreach\bbl@cs@umalqura@data{{\the\count@}}}% month~lunar
\end{verbatim}
This is basically the set of macros written by Michail Rozman in 1991, with corrections and adaptations by Rama Porrat, Misha, Dan Haran and Boris Lavva. This must be eventually replaced by computations with l3fp. An explanation of what's going on can be found in hebcal.sty

12.2 Hebrew

This is basically the set of macros written by Michail Rozman in 1991, with corrections and adaptations by Rama Porrat, Misha, Dan Haran and Boris Lavva. This must be eventually replaced by computations with l3fp. An explanation of what's going on can be found in hebcal.sty
\bbl@hebrelapsedmonths{#1}{#2}\
\tmpa=#2\relax
\multiply \tmpa by 13753
\advance \tmpa by 5604
\bbl@remainder{\tmpa}{25920}{\tmpc}\% \tmpc == ConjunctionParts
\divide \tmpa by 25920
\multiply \tmpa by 29
\advance \tmpa by 1
\advance \tmpa by \tmpa
\bbl@remainder{\tmpa}{7}{\tmpa}\%
\ifnum \tmpc < 19440
\ifnum \tmpc < 9924
\else
\ifnum \tmpa=2
\bbl@checkleaphebryear{#1}\% of a common year
\ifbbl@hebrleap
\else
\advance \tmpa by 1
\fi
\fi
\fi
\ifnum \tmpc < 16789
\else
\ifnum \tmpa=1
\advance \tmpa by -1
\bbl@checkleaphebryear{#1}\% at the end of leap year
\ifbbl@hebrleap
\advance \tmpa by 1
\fi
\fi
\fi
\else
\advance \tmpa by 1
\fi
\fi
\bbl@hebrelapseddays{#1}{\tmpe}\%
\advance \tmpa by 1
\bbl@hebrelapseddays{#1}{#2}\%
\advance \tmpa by -\tmpe
\global\bbl@cntcommon=#2\relax}
#2=\bbl@cntcommon}
\def\bbl@daysinhebyear{#1#2}{%\countdef \tmpe=12
\bbl@hebreldayspriormonths{#1#2}{%\countdef \tmpe=14
#2}{\global \bbl@cntcommon=2}%
#2=\bbl@cntcommon}
\ifnum \the\year > 6
\advance \the\year by 30
\fi
\ifnum \the\year > 3
\ifnum \tmpf=353
\advance \the\year by -1
\fi
\fi
\ifnum \the\year > 2
\ifnum \tmpf=355
\advance \the\year by 1
\fi
\fi
\global\the\year=\the\year
\global\advance \the\year by 3
\the\day=\the\day

\def\absfromgreg#1#2#3#4#5#6{%\countdef\tmpx=17
\countdef\tmpy=18
\countdef\tmpz=19
#6=#3\relax
\the\day=\the\day
\divide \the\year by 30
}
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

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.

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 locallyhyphen.tex or whatever they like, but they mustn't diddle with hyphen.tex (or plain.tex except to preload additional fonts).

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

As these files are going to be read as the first thing ini\TeX{} sees, we need to set some category codes just to be able to change the definition of \input.
If a file called hyphen.cfg can be found, we make sure that it will be read instead of the file hyphen.tex. We do this by first saving the original meaning of \input (and I use a one letter control sequence for that so as not to waste multi-letter control sequence on this in the format).

\begin{verbatim}
\openin 0 hyphen.cfg
\ifeof0
  \let\a\input
\else
  \let\a hyphen.cfg
  \let\a\undefined
\fi
\end{verbatim}

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

\begin{verbatim}
\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}
\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}
\def\fmtname{babel-plain}
\end{verbatim}

When you are using a different format, based on plain.tex you can make a copy of blplain.tex, rename it and replace plain.tex with the name of your format file.

### 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).

\begin{verbatim}
\let\empty{}
\def\loadlocalcfg#1{%
  \openin0#1.cfg
  \ifeof0
    \closein0
  \else
    \closein0
    \immediate\write16{*************************************}%
    \immediate\write16{* Local config file #1.cfg used}%
    \immediate\write16{*}
  \fi
  \input #1.cfg\relax
  \@endofldf}
\end{verbatim}

### 13.3 General tools

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

\begin{verbatim}
\longdef\firstofone#1{#1}
\longdef\firstoftwo#1#2{#1}
\longdef\secondoftwo#1#2{#2}
\def\nil{\@nil}
\def\gobbletwo#1#2{}
\def\ifnextchar*{\ifcsname@firstoftwo#1}}
\end{verbatim}
\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{\relax}
\edef\@backslashchar{\expandafter\@gobble\string\\}
\def\strip@prefix#1>{\@gobble}
\def\g@addto@macro#1#2{{\toks@\expandafter{#1#2}\xdef#1{\the\toks@}}}
\def\@namedef#1{\expandafter\def\csname #1\endcsname}
\def\@nameuse#1{\csname #1\endcsname}
\def\@ifundefined#1{\expandafter\ifx\csname#1\endcsname\relax\expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
\def\@expandtwoargs#1#2#3{\edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
\def\zap@space#1 #2{#1\ifx#2\@empty\else\expandafter\zap@space\fi#2}
\let\bbl@trace\@gobble
\def\bbl@error#1#2{\begingroup
  \newlinechar=`
  \def\\{
  (babel)\ }\errhelp{#2}\errmessage{\#1}\endgroup}
\def\bbl@warning#1{\begingroup
  \newlinechar=`
  \def\\{
  (babel)\message{\#1}\endgroup}
\let\bbl@infowarn\bbl@warning
\def\bbl@info#1{\begingroup
  \newlinechar=`
  \def\\{
  (\wlog{\#1})\endgroup}
\def\begindocument{\@begindocumenthook\global\let\@begindocumenthook\@undefined}
\ifx\@preamblecmds\@undefined
  \def\@preamblecmds{}
\fi
\def\@onlypreamble#1{\expandafter\gdef\expandafter\@preamblecmds\expandafter{\@preamblecmds\do#1}}
\@onlypreamble\@onlypreamble

\LaTeX\ has the command \onlypreamble which adds commands to a list of commands that are no longer needed after \begin{document}.
\ife\preamblecmds\undefined\def\preamblecmds{}\fi
\def\@onlypreamble#1{\expandafter\gdef\expandafter\@preamblecmds\expandafter{\@preamblecmds\do#1}}
\def\@onlypreamble\@onlypreamble

Mimick \LaTeX's \AtBeginDocument; for this to work the user needs to add \begindocument to his file.
\def\begindocument{\@begindocumenthook\global\let\@begindocumenthook\@undefined}
We also have to mimic \LaTeX's \AtEndOfPackage. Our replacement macro is much simpler; it stores its argument in \@endofldf.

\AtEndOfPackage{#1}{\g@addto@macro\@endofldf{#1}}
\@onlypreamble\AtEndOfPackage
\let\@endofldf\@empty
\@onlypreamble\@endofldf

\bbl@afterlang\@empty
\chardef\bbl@opt@hyphenmap\z@

\LaTeX needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default. There is a trick to hide some conditional commands from the outer \ifx. The same trick is applied below.

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

Mimick \LaTeX's commands to define control sequences.

\newcommand{\@star@or@long\new@command}
\def\new@command#1{\@testopt{\@newcommand#1}0}
\def\@newcommand#1[#2]{\@ifnextchar {[\@xargdef#1[#2]}{\@argdef#1[#2]}}
\long\def\@argdef#1[#2]#3{\@yargdef#1\@ne{#2}{#3}}
\long\def\@xargdef#1[#2][#3]#4{\expandafter\def\expandafter#1\expandafter{\expandafter\@protected@testopt\expandafter #1\csname\string#1\endcsname{#3}}\expandafter\@yargdef\csname\string#1\endcsname\tw@{#2}{#4}}
\long\def\@yargdef#1#2#3{\@tempcnta#3\relax\advance\@tempcnta\@ne\let\@hash@\relax\edef\reserved@a{\ifx#2\tw@\@hash@1\fi}\@tempcntb#2\@whilenum\@tempcntb<\@tempcnta\do{\edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}}\let\@hash@##\l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
The following little macro is taken from `latex.ltx`; it checks whether its first argument is part of its second argument. It uses the boolean \in@; allocating a new boolean inside conditionally executed code is not possible; hence the construct with the temporary definition of \bbl@tempa.

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

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

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

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

\def\@ifl@aded#1#2#3#4{}
To prevent wasting two counters in \TeX (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (\count10).

\ifx\bye\@undefined
\advance\count10 by -2\relax
\fi
\ifx\@ifnextchar\@undefined
\def\@ifnextchar#1#2#3{\
\let\reserved@d=#1\%
\def\reserved@a{#2}\def\reserved@b{#3}\%
\futurelet\@let@token\@ifnch}
\def\@ifnch{\
\ifx\@let@token\@sptoken
\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=} 
\def\:{\@xifnch} \expandafter\def\:\expandafter{\futurelet\@let@token\@ifnch}
\fi
\ifx\@testopt\#1\#2\%
\@ifnextchar\[{\#1}{\#1\[#2\]}\%
\def\@protected@testopt#1\%
\ifx\protect\@typeset@protect
\expandafter\@testopt
\else
\@x@protect#1\%
\fi\}
\long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax #2\relax}\fi}
\long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
\else\expandafter\@gobble\fi{#1}}

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}}
\def\@current@cmd#1{\ifx\protect\@typeset@protect\else
\noexpand#1\expandafter\@gobble\fi{#1}}
Currently we only use the \text{H\textsuperscript{H}P\textsubscript{X}} method for accents for those that are known to be made active in some language definition file.

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

For a couple of languages we need the \text{H\textsuperscript{H}P\textsubscript{X}}-control sequence \texttt{\scriptsize} to be available. Because plain \TeX\ doesn’t have such a sophisticated font mechanism as \text{H\textsuperscript{H}P\textsubscript{X}} has, we just \texttt{\let\scriptsize\sevenrm}.

And a few more “dummy” definitions.
14 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