Babel

Localization and internationalization

Unicde
TEx
pdfTEx
LuaTEx
XeTEx
# Contents

1 Identification and loading of required files 3

2 locale directory 3

3 Tools 3

3.1 Multiple languages 7

3.2 The Package File (\LaTeX{}, babel.sty) 8

3.3 base 9

3.4 key=value options and other general option 10

3.5 Conditional loading of shorthands 11

3.6 Interlude for Plain 13

4 Multiple languages 13

4.1 Selecting the language 15

4.2 Errors 23

4.3 Hooks 25

4.4 Setting up language files 27

4.5 Shorthands 29

4.6 Language attributes 38

4.7 Support for saving macro definitions 39

4.8 Short tags 41

4.9 Hyphens 41

4.10 Multiencoding strings 43

4.11 Macros common to a number of languages 48

4.12 Making glyphs available 48

4.12.1 Quotation marks 48

4.12.2 Letters 50

4.12.3 Shorthands for quotation marks 50

4.12.4 Umlauts and tremas 51

4.13 Layout 52

4.14 Load engine specific macros 53

4.15 Creating and modifying languages 53

5 Adjusting the Babel behavior 76

5.1 Cross referencing macros 79

5.2 Marks 81

5.3 Preventing clashes with other packages 82

5.3.1 ifthen 82

5.3.2 varioref 83

5.3.3 hhline 83

5.4 Encoding and fonts 84

5.5 Basic bidi support 85

5.6 Local Language Configuration 89

5.7 Language options 89

6 The kernel of Babel (babel.def, common) 92

7 Loading hyphenation patterns 96

8 Font handling with fontspec 100

9 Hooks for XeTeX and LuaTeX 103

9.1 XeTeX 103
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Support for interchar</td>
<td>105</td>
</tr>
<tr>
<td>10.1</td>
<td>Layout</td>
<td>107</td>
</tr>
<tr>
<td>10.2</td>
<td>8-bit TeX</td>
<td>109</td>
</tr>
<tr>
<td>10.3</td>
<td>LuaTeX</td>
<td>110</td>
</tr>
<tr>
<td>10.4</td>
<td>Southeast Asian scripts</td>
<td>116</td>
</tr>
<tr>
<td>10.5</td>
<td>CJK line breaking</td>
<td>117</td>
</tr>
<tr>
<td>10.6</td>
<td>Arabic justification</td>
<td>119</td>
</tr>
<tr>
<td>10.7</td>
<td>Common stuff</td>
<td>124</td>
</tr>
<tr>
<td>10.8</td>
<td>Automatic fonts and ids switching</td>
<td>124</td>
</tr>
<tr>
<td>10.9</td>
<td>Bidi</td>
<td>130</td>
</tr>
<tr>
<td>10.10</td>
<td>Layout</td>
<td>132</td>
</tr>
<tr>
<td>10.11</td>
<td>Lua: transforms</td>
<td>139</td>
</tr>
<tr>
<td>10.12</td>
<td>Lua: Auto bidi with basic and basic-r</td>
<td>149</td>
</tr>
<tr>
<td>11</td>
<td>Data for CJK</td>
<td>160</td>
</tr>
<tr>
<td>12</td>
<td>The 'nil' language</td>
<td>160</td>
</tr>
<tr>
<td>13</td>
<td>Calendars</td>
<td>161</td>
</tr>
<tr>
<td>13.1</td>
<td>Islamic</td>
<td>161</td>
</tr>
<tr>
<td>13.2</td>
<td>Hebrew</td>
<td>163</td>
</tr>
<tr>
<td>13.3</td>
<td>Persian</td>
<td>167</td>
</tr>
<tr>
<td>13.4</td>
<td>Coptic and Ethiopic</td>
<td>167</td>
</tr>
<tr>
<td>13.5</td>
<td>Buddhist</td>
<td>168</td>
</tr>
<tr>
<td>14</td>
<td>Support for Plain TeX (plain.def)</td>
<td>169</td>
</tr>
<tr>
<td>14.1</td>
<td>Not renaming hyphen.tex</td>
<td>169</td>
</tr>
<tr>
<td>14.2</td>
<td>Emulating some EfP\text features</td>
<td>170</td>
</tr>
<tr>
<td>14.3</td>
<td>General tools</td>
<td>171</td>
</tr>
<tr>
<td>14.4</td>
<td>Encoding related macros</td>
<td>174</td>
</tr>
<tr>
<td>15</td>
<td>Acknowledgements</td>
<td>177</td>
</tr>
</tbody>
</table>
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 L\TeX package, which set options and load language styles.
- `babel.def` is loaded by Plain.
- `switch.def` defines macros to set and switch languages (it loads part `babel.def`).
- `plain.def` is not used, and just loads `babel.def`, for compatibility.
- `hyphen.cfg` is the file to be used when generating the formats to load hyphenation patterns.

There are some additional tex, def and lua files

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

2 locale directory

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

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

See Keys in ini files in the the babel site.

3 Tools

```
1 ⟨⟨version=24.6⟩⟩
2 ⟨⟨date=2024/06/02⟩⟩

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

We define some basic macros which just make the code cleaner. \bbl@add is now used internally instead of \addto because of the unpredictable behavior of the latter. Used in `babel.def` and in `babel.sty`, which means in \LaTeX is executed twice, but we need them when defining options and `babel.def` cannot be load until options have been defined. This does not hurt, but should be fixed somehow.
```
\def\bbl@loop#1#2#3{\bbl@@loop#1{#3}#2,\@nnil,}
\def\bbl@loopx#1#2{\expandafter\bbl@loop\expandafter#1\expandafter{#2}}
\def\bbl@@loop#1#2#3,{%
  \ifx\@nnil#3\relax\else
  \def#1{#3}#2\bbl@afterfi\bbl@@loop#1{#2}%
  \fi}
\def\bbl@for#1#2#3{\bbl@loopx#1{#2}{\ifx#1\@empty\else#3\fi}}
\bbl@add@list
This internal macro adds its second argument to a comma separated list in its first argument. When the list is not defined yet (or empty), it will be initiated. It presumes expandable character strings.
\def\bbl@add@list#1#2{%
  \edef#1{\bbl@ifunset{\bbl@stripslash#1}{}{\ifx#1\@empty\else#1,\fi}#2}}
\bbl@afterelse
Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the \else and \fi parts of an \if-statement\footnote{This code is based on code presented in TUGboat vol. 12, no2, June 1991 in “An expansion Power Lemma” by Sonja Maus.}. These macros will break if another \if...i statement appears in one of the arguments and it is not enclosed in braces.
\def\bbl@afterelse#1\else#2\fi{\fi#1}
\def\bbl@afterfi#1\fi{\fi#1}
\bbl@exp
Now, just syntactical sugar, but it makes partial expansion of some code a lot more simple and readable. Here \s stands for \noexpand, \..<..> for \noexpand applied to a built macro name (which does not define the macro if undefined to \relax, because it is created locally), and \[..] for one-level expansion (where .. is the macro name without the backslash). The result may be followed by extra arguments, if necessary.
\def\bbl@exp##1##2{\begingroup\let\\noexpand\let<\bbl@exp@en\let\[]\bbl@exp@ue\edef\bbl@exp@aux{\endgroup##1}\bbl@exp@aux}
\def\bbl@exp@en#1>{\expandafter\noexpand\csname#1\endcsname}
\def\bbl@exp@ue#1\}{\unexpanded\expandafter\expandafter\expandafter{\csname#1\endcsname}}
\bbl@trim
The following piece of code is stolen (with some changes) from keyval, by David Carlisle. It defines two macros: \bbl@trim and \bbl@trimdef. The first one strips the leading and trailing spaces from the second argument and then applies the first argument (a macro, \toks@ and the like). The second one, as its name suggests, defines the first argument as the stripped second argument.
\def\bbl@tempa#1{\def\bbl@trim##1##2{\futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}\def\bbl@trim@c{\ifx\bbl@trim@a\@sptoken\bbl@trim@b\else\bbl@trim@b\expandafter#1\fi}}\def\bbl@trim@b#1##1\@nil{\bbl@trim@i##1}\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}\def\bbl@trim@def#1{\bbl@trim\def#1}
\bbl@ifunset
To check if a macro is defined, we create a new macro, which does the same as \@ifundefined. However, in an \epsilon-tex engine, it is based on \ifcsname, which is more efficient, and does not waste
memory. Defined inside a group, to avoid \ifcsname being implicitly set to \relax by the \csname test.

\begingroup
\ifcsname
\expandafter\ifx\csname#1\endcsname\relax
\else
\expandafter\@firstoftwo
\fi
\endgroup
\ifcsname
\expandafter\ifx\csname#1\endcsname\relax
\bbl@afterelse\expandafter\@firstoftwo
\else
\bbl@afterfi\expandafter\@secondoftwo
\fi
\endgroup
\begin{itemize}
\item \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, i.e., not \relax and not empty,
\end{itemize}

\begin{itemize}
\item \bbl@forkv
\end{itemize}

\begin{itemize}
\item \bbl@foreach
\end{itemize}

\begin{itemize}
\item \bbl@replace
\end{itemize}
An extension to the previous macro. It takes into account the parameters, and it is string based (i.e., if you replace \relax by ho, then \relax becomes \rho). No checking is done at all, because it is not a general purpose macro, and it is used by babel only when it works (an example where it does not work is in \bbl@TG@date, and also fails if there are macros with spaces, because they are retokenized). It may change! (or even merged with \bbl@replace; I’m not sure checking the replacement is really necessary or just paranoia).

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

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

The following adds some code to \extras... both before and after, while avoiding doing it twice. It's somewhat convoluted, to deal with #'s. Used to deal with alph, Alph and french spacing 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 \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. The following block is used in switch . def and hyphen . cfg; the latter may seem redundant, but remember babel doesn't requires loading switch . def in the format.
Another counter is used to keep track of the allocated languages. \TeX{} and \LaTeX{} reserves for this purpose the count 19.

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

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

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

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

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

```latex
\@ifpackagewith{babel}{debug}{
  \providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
  \let\bbl@debug\@firstofone
  \ifx\directlua\@undefined\else
    \directlua{ Babel = Babel or {};
      Babel.debug = true }
    \input{babel-debug.tex}
  \fi
}
\def\bbl@error#1{%
  \begingroup
    \catcode`\=0 \catcode`\!=12 \catcode`\!=12
  \input errbabel.def
  \endgroup
  \bbl@error{#1}}
\def\bbl@warning#1{%
  \begingroup
    \def\{\MessageBreak
    \PackageWarning{babel}{#1}
  \endgroup}
\def\bbl@infowarn#1{%
  \begingroup
    \def\{\MessageBreak
    \PackageNote{babel}{#1}
  \endgroup}
\def\bbl@info#1{%
  \begingroup
    \def\{\MessageBreak
    \PackageInfo{babel}{#1}
  \endgroup
```

8
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.

\@ifpackagewith{babel}{showlanguages}{%
\begingroup
\catcode`\^^I=12
\def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
\wlog{<*languages>}\bbl@languages\wlog{</languages>}%
\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 \texxforgets 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 \latex) hyphenation patterns, even if we are not interested in the rest of babel.

\bbl@trace{Defining option ‘base’}
\@ifpackagewith{babel}{base}{%
\begingroup
\let\bbl@onlyswitch\@empty
\let\bbl@provide@locale\relax
\input babel.def
\let\bbl@onlyswitch\@undefined
\ifx\directlua\@undefined\DeclareOption*{\bbl@patterns{\CurrentOption}}%
\else\input luababel.def\DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
\fi%
\bbl@languages\f\}

\@ifpackagewith{babel}{empty}{%}
\@ifpackagewith{babel}{relax}{%}
\input babel.def
\let\bbl@onlyswitch\undefined
\ifx\directlua\undefined
\DeclareOption*{\bbl@patterns{\CurrentOption}}%
\else
\input luababel.def
\DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
\fi%}
\ProcessOptions
\global\expandafter\let\csname opt@babel.sty\endcsname\relax
\global\expandafter\let\csname ver@babel.sty\endcsname\relax
\global\let\@ifl@ter@@\@ifl@ter
\def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
3.4  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.

\begininput{}

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

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

\begin{verbatim}
\let\bbl@opt@shorthands\@nil
\let\bbl@opt@config\@nil
\let\bbl@opt@main\@nil
\let\bbl@opt@headfoot\@nil
\let\bbl@opt@layout\@nil
\let\bbl@opt@provide\@nil
\end{verbatim}

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

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

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

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

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

\begin{verbatim}
\ProcessOptions*
\let\bbl@opt@provide\@nil
\let\bbl@opt@provide\@empty % % MOVE above
\else
  \chardef\bbl@iniflag\@ne
  \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%\CurrentOption\bbl@tempa}
  \\\}
  \bbl@replace\bbl@opt@provide{;}{,}{}
\fi
\end{verbatim}

\subsection*{3.5 Conditional loading of shorthands}

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

```latex
\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).

```latex\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.

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

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

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

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

```
\def\bbl@opt@safe\undefined
\def\bbl@opt@safe{BR}
% \let\bbl@opt@safe\@empty % Pending of \cite
```

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

```
\bbl@exp{\newcommand{\IfBabelLayout}[3]{\@make@pass\IfBabelLayout\def\bbl@opt@layout\@nil\let\protect\noexpand}}
\bbl@exp{\newcommand{\IfBabelLayout}[1]{\@make@pass\IfBabelLayout\def\bbl@opt@layout\@nil\let\protect\noexpand}}
```

⟨/package⟩
3.6 Interlude for Plain

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

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

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

```
⟨/core⟩
⟨∗package | core⟩
```

4 Multiple languages

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

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

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

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

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

```
\def\bbl@iflanguage#1{%
\begingroup
\def\bbl@tempe{l@}\
\edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}\
\bbl@tempd\
\fi}\
\bbl@cs{languages}\
\endgroup}
```

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

```
\def\bbl@iflanguage#1{%
\begingroup
\def\bbl@tempe{l@}\
\edef\bbl@temped{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}\
\bbl@temped\
\ifnum\count@=##2\relax
\edef\bbl@tempa{\expandafter\@gobbletwo\string#1}\
\bbl@info{Hyphen rules for \expandafter\string\csname l@##1\endcsname\%
\string\language\the\count@}. Reported}\
\bbl@elt####1####2####3####4{}%}
\bbl@cs{languages}\
\endgroup}
```

13
After a name has been ‘fixed’, the selectors will try to load the language. If even the fixed name is not defined, 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 BCP47 lookup. It also makes sure, with \bbl@bcp-case, 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@bcp-lookup either returns the found ini or is \relax.

\def\bbl@bcp-case#1#2#3#4\@@#5{%  
  \ifx\empty#3%  
    \uppercase{\def#5{#1#2}}%  
  \else%  
    \uppercase{\def#5{#1}}%  
    \lowercase{\edef#5{#5#2#3#4}}%  
  \fi%  
}\def\bbl@bcp-lookup#1-#2-#3-#4\@@{%  
  \let\bbl@bcp\relax  
  \lowercase{\def\bbl@tempa{#1}}%  
  \ifx\empty#2%  
    \IfFileExists{babel-bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%  
  \else\ifx\empty#3%  
    \bbl@bcp-case#2\empty\empty\@@\bbl@tempb  
    \IfFileExists{babel-bbl@tempa-bbl@tempb.ini}{}%  
  \fi%  
  \ifx\bbl@bcp\relax  
    \IfFileExists{babel-bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%  
  \fi%  
  \else%  
    \bbl@bcp-case#2\empty\empty\@@\bbl@tempb  
    \bbl@bcp-case#3\empty\empty\@@\bbl@tempc  
    \IfFileExists{babel-bbl@tempa-bbl@tempb-bbl@tempc.ini}{}%  
  \fi%  
  \ifx\bbl@bcp\relax  
    \IfFileExists{babel-bbl@tempa-bbl@tempc.ini}{\let\bbl@bcp\bbl@tempa-bbl@tempc}{}%  
  \fi%  
  \ifx\bbl@bcp\relax  
    \IfFileExists{babel-bbl@tempa-bbl@tempc.ini}{\let\bbl@bcp\bbl@tempa-bbl@tempc}{}%  
  \fi%  
  \ifx\bbl@bcp\relax  
    \IfFileExists{babel-bbl@tempa-bbl@tempc.ini}{\let\bbl@bcp\bbl@tempa-bbl@tempc}{}%  
  \fi%  
  \ifx\bbl@bcp\relax  
    \IfFileExists{babel-bbl@tempa-bbl@tempc.ini}{\let\bbl@bcp\bbl@tempa-bbl@tempc}{}%  
  \fi%  
  \let\bbl@initoload\relax  
⟨-core⟩  
\def\bbl@provide@locale{%  
  \ifx\babelprovide\@undefined%  
    \bbl@error{base-on-the-fly}{}{}{}%  
  \fi%  
\def\bbl@numerical-coding{%  
  \ifx\babelnumerical-coding\@undefined%  
    \bbl@error{base-on-the-fly}{}{}{}%  
  \fi%  
\def\bbl@bcp-case#1#2#3#4\@@#5{%  
  \ifx\empty#3%  
    \uppercase{\def#5{#1#2}}%  
  \else%  
    \uppercase{\def#5{#1}}%  
    \lowercase{\edef#5{#5#2#3#4}}%  
  \fi%  
}\def\bbl@bcp-lookup#1-#2-#3-#4\@@{%  
  \let\bbl@bcp\relax  
  \lowercase{\def\bbl@tempa{#1}}%  
  \ifx\empty#2%  
    \IfFileExists{babel-bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%  
  \else\ifx\empty#3%  
    \bbl@bcp-case#2\empty\empty\@@\bbl@tempb  
    \IfFileExists{babel-bbl@tempa-bbl@tempb.ini}{}%  
  \fi%  
  \ifx\bbl@bcp\relax  
    \IfFileExists{babel-bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%  
  \fi%  
  \else%  
    \bbl@bcp-case#2\empty\empty\@@\bbl@tempb  
    \bbl@bcp-case#3\empty\empty\@@\bbl@tempc  
    \IfFileExists{babel-bbl@tempa-bbl@tempb-bbl@tempc.ini}{}%  
  \fi%  
  \ifx\bbl@bcp\relax  
    \IfFileExists{babel-bbl@tempa-bbl@tempc.ini}{\let\bbl@bcp\bbl@tempa-bbl@tempc}{}%  
  \fi%  
  \ifx\bbl@bcp\relax  
    \IfFileExists{babel-bbl@tempa-bbl@tempc.ini}{\let\bbl@bcp\bbl@tempa-bbl@tempc}{}%  
  \fi%  
  \ifx\bbl@bcp\relax  
    \IfFileExists{babel-bbl@tempa-bbl@tempc.ini}{\let\bbl@bcp\bbl@tempa-bbl@tempc}{}%  
  \fi%  
  \ifx\bbl@bcp\relax  
    \IfFileExists{babel-bbl@tempa-bbl@tempc.ini}{\let\bbl@bcp\bbl@tempa-bbl@tempc}{}%  
  \fi%  
  \ifx\bbl@bcp\relax  
    \IfFileExists{babel-bbl@tempa-bbl@tempc.ini}{\let\bbl@bcp\bbl@tempa-bbl@tempc}{}%  
  \fi%  
  \ifx\bbl@bcp\relax  
    \IfFileExists{babel-bbl@tempa-bbl@tempc.ini}{\let\bbl@bcp\bbl@tempa-bbl@tempc}{}%  
  \fi%  
  \let\bbl@auxname\languagename % Still necessary. TODD
\def\bbl@ifunset{\bbl@bcp-map\languagename}{% Move uplevel??
  \edef\languagename{/nameuse{\bbl@bcp-map\languagename}}}%
\iflanguage

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

4.1 Selecting the language

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.

\let\bbl@select@type\z@
\edef\selectlanguage{%
\noexpand\protect
\expanded\noexpand\csname selectlanguage \endcsname}

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.

\ifx\undefined\protect\let\protect\relax\fi

The following definition is preserved for backwards compatibility (eg, arabic, 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.

But when the language change happens inside a group the end of the group doesn’t write anything to the auxiliary files. Therefore we need \TeX’s \aftergroup mechanism to help us. The command \aftergroup stores the token immediately following it to be executed when the current group is closed. So we define a temporary control sequence \bbl@pop@language to be executed at the end of the group. It calls \bbl@set@language with the name of the current language as its argument.
The previous solution works for one level of nesting groups, but as soon as more levels are used it is no longer adequate. For that case we need to keep track of the nested languages using a stack mechanism. This stack is called `\bbl@language@stack` and initially empty.

```
\def\bbl@language@stack{}
```

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

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

```
\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
```

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

```
\def\bbl@pop@lang#1+#2\@@{\edef\languagename{#1}\xdef\bbl@language@stack{#2}}
```

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

```
\let\bbl@ifrestoring\@secondoftwo
\def\bbl@pop@language{\
  \expandafter\bbl@pop@lang\bbl@language@stack\@@
  \let\bbl@ifrestoring\@firstoftwo
  \expandafter\bbl@set@language\expandafter{\languagename}
\}
```

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

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

```
\chardef\localeid\z@
\def\bbl@id@assign{\bbl@ifunset{bbl@id@@\languagename}{\count@\bbl@id@last\relax\
  \addtocounter{\@name}{\count@}\bbl@csarg\chardef{id@@\languagename}\count@
  \ifcase\bbl@engine\or\
    \directlua{
      Babel = Babel or {}\
      Babel.locale_props = Babel.locale_props or {}\
      Babel.locale_props[\bbl@id@last] = {}\
      Babel.locale_props[\bbl@id@last].name = '\languagename'
    }
  }\fi}
```
The unprotected part of \selectlanguage. In case it is used as environment, declare \endselectlanguage, just for safety.
\begin{verbatim}
\expandafter\def\csname selectlanguage \endcsname#1{%
\ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@\fi
\bbl@push@language
\aftergroup\bbl@pop@language
\bbl@set@language{#1}}
\let\endselectlanguage\relax
\bbl@set@language

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

We also write a command to change the current language in the auxiliary files. \bbl@savelastskip is used to deal with skips before the write whatsit (as suggested by U Fischer). Adapted from hyperref, but it might fail, so I’ll consider it a temporary hack, while I study other options (the ideal, but very likely unfeasible except perhaps in luatex, is to avoid the \write altogether when not needed).
\begin{verbatim}
\def\BabelContentsFiles{toc,lof,lot}
\def\bbl@set@language#1{% from selectlanguage, pop@
% The old buggy way. Preserved for compatibility.
\edef\languagename{\ifnum\escapechar=\expandafter`\string#1\@empty\else\string#1\@empty\fi}
\ifcat\relax
\expandafter\ifx\csname date\languagename\endcsname\relax
\edef\languagename{#1}
\let\localename\languagename
\else
\bbl@info{Using \string\language instead of language is\%
deprecated. If what you want is to use a\%
macro containing the actual locale, make\%
sure it does not not match any language.\%
Reported}\%
\if\scantokens\@undefined
\def\localename{??}\else
\scantokens\expandafter{\expandafter\def\expandafter\localename\expandafter{\languagename}}\fi
\else
\bb@info{Using \string\language instead of language is\%
deprecated. If what you want is to use a\%
macro containing the actual locale, make\%
sure it does not not match any language.\%
Reported}\%
\if\scantokens\undefined
\def\localename{??}\else
\scantokens\expandafter{\expandafter\def\expandafter\localename\expandafter{\languagename}}\fi
\fi
\fi
\else
\def\localename{#1}\% This one has the correct catcodes
\fi
\select@language{\languagename}
% write to auxs
\expandafter\ifx\csname date\languagename\endcsname\relax\else
\expandafter\if\filesw
\if\bbabel@aux\@gobble\fi\else % Set if single in the first, redundant
\bbl@savelastskip
\protected@write\@auxout{}{\string\babel@aux{\bbl@auxname}{}}%
\bbl@restorelastskip
\fi
\fi
\bbl@usehooks{write}{%
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 \langhyphenmins is defined. If it is not, we set default values (2 and 3), otherwise the values in \langhyphenmins will be used. No text is supposed to be added with switching captions and date, so we remove any spurious spaces with \bsphack and \esphack.
otherlanguage (em.) The otherlanguage environment can be used as an alternative to using the \selectlanguage declarative command. The \ignorespaces command is necessary to hide the environment when it is entered in horizontal mode.

\long\def\otherlanguage#1{\def\bbl@selectorname{other}\ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi\csname selectlanguage \endcsname{#1}\ignorespaces}

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

\long\def\endotherlanguage{\@ignoretrue\ignorespaces}

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

\def\csname otherlanguage*\endcsname{\@ifnextchar\[\bbl@otherlanguage@s{\bbl@otherlanguage@s[\]}}\def\bbl@otherlanguage@s[#1]{\def\bbl@selectorname{other*}\ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi\def\bbl@select@opts{#1}\foreignlanguage{#2}}

At the end of the environment we need to switch off the extra definitions. The grouping mechanism of the environment will take care of resetting the correct hyphenation rules and “extras”.

\expandafter\let\csname endotherlanguage*\endcsname\relax
\providecommand\bbl@beforeforeign{}
\edef\foreignlanguage{\noexpand\protect\expandafter\def\csname foreignlanguage \endcsname}
\def\bbl@select@opts{#1}%
\let\BabelText@firstofone
\foreign@language{#2}%
\bbl@usehooks{foreign}{}
\BabelText{#3}% Now in horizontal mode!
\endgroup}
\def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
\begingroup
{\par}
\def\bbl@selectorname{foreign*}%
\let\bbl@select@opts\@empty
\let\BabelText@firstofone
\foreign@language{#1}%
\bbl@usehooks{foreign*}{}
\bbl@dirparastext
\BabelText{#2}% Still in vertical mode!
{\par}
\endgroup}

\foreign@language
This macro does the work for \foreign@language and the other language* environment. First we need to store the name of the language and check that it is a known language. Then it just calls bbl@switch.
\def\foreign@language#1{%
% set name
\edef\languagename{#1}%
\ifbbl@usedategroup
\bbl@add\bbl@select@opts{,date,}%
\bbl@usedategroupfalse
\fi
% TODO. name@map here?
\bbl@provide@locale
\bbl@iflanguage\languagename{%
\let\bbl@select@type\@ne
\expandafter\bbl@switch\expandafter{\languagename}}

The following macro executes conditionally some code based on the selector being used.
\def\IfBabelSelectorTF#1{%
\bbl@xin@{,\bbl@selectorname,}{,\zap@space#1 \@empty,}%
\ifin@
\expandafter\@firstoftwo
\else
\expandafter\@secondoftwo
\fi}

\bbl@patterns
This macro selects the hyphenation patterns by changing the \language register. If special hyphenation patterns are available specifically for the current font encoding, use them instead of the default. It also sets hyphenation exceptions, but only once, because they are global (here language \lccode's has been set, too). \bbl@hyphenation@ is set to relax until the very first \babel@hyphenation, so do nothing with this value. If the exceptions for a language (by its number, not its name, so that :NC is taken into account) has been set, then use \hyphenation with both global and language exceptions and empty the latter to mark they must not be set again.
\let\bbl@hyphlist\@empty
\let\bbl@hyphenation@\relax
\let\bbl@pttnlist\@empty
\let\bbl@patterns@\relax
\let\bbl@hmapsel@\@cclv
\def\bbl@patterns#1{%
\language=\expandafter\ifx\csname l@#1:f@encoding\endcsname\relax
\csname l@#1:endcsname
\edef\bbl@tempa{#1}%
\def\hyphenrules#1{\edef\bbl@tempf{#1}\bbl@fixname\bbl@tempf\bbl@iflanguage\bbl@tempf{%\expandafter\bbl@patterns\expandafter{\bbl@tempf}\ifx\languageshorthands\@undefined\else\languageshorthands{none}\fi\expandafter\expandafter\expandafter\set@hyphenmins\csname\bbl@tempf hyphenmins\endcsname\relax\set@hyphenmins\tw@\thr@@\relax\else\expandafter\expandafter\expandafter\set@hyphenmins\csname\bbl@tempf hyphenmins\endcsname\relax\fi\set@hyphenmins\tw@\thr@@\relax\let\endhyphenrules\@empty\providehyphenmins\set@hyphenmins

\def\providehyphenmins#1#2{\expandafter\ifx\csname#1hyphenmins\endcsname\relax\@namedef{#1hyphenmins}{#2}\fi}

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

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

\begin{verbatim}
835 \else
836     \csname l@#1:\f@encoding\endcsname
837 \edef\bbl@tempa{#1:\f@encoding}\
838 \fi
839 \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
840 % > luatex
841 \@ifundefined{bbl@hyphenation@}{\relax}{{\begingroup
842 \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
843 \ifin@\else
844 \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
845 \hyphenation{\bbl@hyphenation@}
846 \if@empty
847 \makebox[0pt]{\space}\csname bbl@hyphenation@\endcsname
848 \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
849 \fi
850 \endgroup}}

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

\def\providehyphenmins#1#2{\expandafter\ifx\csname#1hyphenmins\endcsname\relax\@namedef{#1hyphenmins}{#2}\fi}

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

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

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

\providecommand\setlocale{bl@error{not-yet-available}{}{}{}}
\let\uselocale\setlocale
\let\locale\setlocale
\let\selectlocale\setlocale
\let\textlocale\setlocale
\let\textlanguage\setlocale
\let\languagetext\setlocale

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

\@noopterr
When the package was loaded without options not everything will work as expected. An error message is issued in that case.

When the format knows about \PackageError it must be \TeX\LaTeX\epsilon, so we can safely use its error handling interface. Otherwise we’ll have to ‘keep it simple’. Infos are not written to the console, but on the other hand many people think warnings are errors, so a further message type is defined: an important info which is sent to the console.

\edef\bbl@nulllanguage{\string\language=0}
\def\bbl@nocaption{\protect\bbl@nocaption@i}
\def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
\global\@namedef{#2}{\textbf{?#1?}}\@nameuse{#2} \edef\bbl@tempa{#1} \bbl@sreplace\bbl@tempa{name}{} \bbl@warning{\@backslashchar#1 not set for ‘\languagename’. \textbf{Please,}} \textbf{define it after the language has been loaded} \textbf{(typically in the preamble) with:\string\setlocalecaption{\languagename}{\bbl@tempa}{..} \textbf{Feel free to contribute on github.com/latex3/babel. \textbf{Reported}}}
\def\bbl@tentative{\protect\bbl@tentative@i}
\def\bbl@tentative@i#1{\bbl@warning{\@backslashchar#1 not set for ‘\languagename’. \textbf{Please,}} \textbf{define it after the language has been loaded} \textbf{(typically in the preamble) with:\string\setlocalecaption{\languagename}{\bbl@tempa}{..} \textbf{Feel free to contribute on github.com/latex3/babel. \textbf{Reported}}}
\def\molanerr#1{\bbl@error{undefined-language}{#1}{}{}}
\def\nopatterns#1{\bbl@warning{No hyphenation patterns were preloaded for the language '#1' into the format. Please, configure your TeX system to add them and rebuild the format. Now I will use the patterns preloaded for \bbl@nulllanguage\space instead}}
\let\bbl@usehooks\@gobbletwo
\ifx\bbl@onlyswitch\@empty\endinput\fi
% Here ended switch.def
\addto\Itakestwoarguments,a\langle\controlsequence\rangle\text{and}\text{\LaTeX}-\text{code to be added to the }\langle\controlsequence\rangle.\text{If the }\langle\controlsequence\rangle\text{has not been defined before it is defined now. The control sequence could also expand to }\text{\LaTeX},\text{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.}\addto\Itakestwoarguments,a\langle\controlsequence\rangle\text{and}\text{\LaTeX}-\text{code to be added to the }\langle\controlsequence\rangle.\text{If the }\langle\controlsequence\rangle\text{has not been defined before it is defined now. The control sequence could also expand to }\text{\LaTeX},\text{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.}
\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 \texttt{\macro} will be saved new control sequences named \texttt{\org@macro}.

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

\bbl@redefine@long

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

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

\bbl@redefinerobust

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

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

4.3 Hooks

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

\begin{verbatim}
\def\bbl@usehooks{%
\def\bbl@usehooks@lang#1#2#3{%
  \ifx\UseHook\@undefined\else\UseHook{babel/*/#2}\fi
  \def\bbl@elth##1{\bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#1}#3}}%
  \bbl@cs{ev@#2@#1}%
  \ifx\languagename\@undefined\else % Test required for Plain (?)
    \ifx\UseHook\@undefined\else\UseHook{babel/#1/#2}\fi
    \def\bbl@elth##1{\bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#1@#2}#3}}%
    \bbl@cs{ev@#2@#1@#2}\fi
}\end{verbatim}
To ensure forward compatibility, arguments in hooks are set implicitly. So, if a further argument is added in the future, there is no need to change the existing code. Note events intended for hyphen.cfg are also loaded (just in case you need them for some reason).

```latex
1016 \def\bbl@evargs{,% <- don't delete this comma
1017 \everylanguage=1, \loadkernel=1, \loadpatterns=1, \loadexceptions=1, %
1018 \adddialect=2, \patterns=2, \defaultcommands=0, \encodedcommands=2, \write=0, %
1019 \beforeextras=0, \afterextras=0, \stopcommands=0, \stringprocess=0, %
1020 \hyphenation=2, \initiateactive=3, \afterreset=0, \foreign=0, \foreign*=0, %
1021 \beforestart=0, \languagename=2, \begindocument=1}
```

The user command just parses the optional argument and creates a new macro named \bbl@ensure\langle 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 language\rangle contains \bbl@ensure\langle include\rangle\langle exclude\rangle\langle fontenc\rangle, which in in turn loops over the macros names in \bbl@captionslist, excluding (with the help of \in\langle\rangle\langle lang\rangle\langle include\rangle\langle exclude\rangle\langle fontenc\rangle in the exclude list. If the fontenc is given (and not $\relax$), the \fontencoding is also added. Then we loop over the include list, but if the macro already contains \foreignlanguage, nothing is done. Note this macro (1) is not restricted to the preamble, and (2) changes are local.

```latex
1026 \bbl@trace{Defining babelensure}
1027 \newcommand\babelensure[2][2][2][]{%
1028 \AddBabelHook{babel-ensure}{afterextras}{%\bbl@select@type\bbl@cl{e}%;}%
1029 \begingroup
1030 \let\bbl@ens@include\@empty
1031 \let\bbl@ens@exclude\@empty
1032 \def\bbl@ens@fontenc{\relax}%
1033 \def\bbl@tempb##1{%
1034 \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1035 \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1036 \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ens@##1}{##2}}%
1037 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
1038 \def\bbl@tempc{\bbl@ensure}%
1039 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{\
1040 \expandafter{\bbl@include}}%
1041 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{\
1042 \expandafter{\bbl@exclude}}%
1043 \toks@\expandafter{\bbl@tempc}%
1044 \bbl@exp{%
1045 \endgroup
1046 \bbl@include\langle#2\rangle}{%the\toks@{\bbl@include\langle#1\rangle}}}
1047 \edef\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1048 \edef\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1049 \ifx\@empty##1\else\noexpand\bbl@tempslash\bbl@tempb\fi}%
1050 \edef\bbl@tempslash##1{\@def\bbl@tempslash{##1}{##1}{##1}}%
1051 \edef\bbl@tempslash##1{\@def\bbl@tempslash{##1}{##1}{##1}}%
1052 \ifx##1\undefined % 3.32 - Don't assume the macro exists
1053 \edef##1{\noexpand\\noexpand\bbl@tempslash{##1}{\bbl@tempslash{##1}{##1}}}%
1054 \bbl@noset{\bbl@tempslash{##1}{\bbl@tempslash{##1}{##1}}}%
1055 \fi
1056 \bbl@ifempty{\ifx##1\empty\else
1057 \in\langle#1\rangle\langle#2\rangle%
1058 \in\langle\rangle\langle#2\rangle%
1059 \bbl@ifunset{\bbl@ensure\langle\rangle\langle#1\rangle\langle\rangle\langle#2\rangle}%
1060 \\DeclareRobustCommand\bbl@ensure\langle#1\rangle[1]\langle#1\rangle{%
1061 \\\foreignlanguage\langle#1\rangle{%
1062 \\\relax%\fontencoding\langle#3\rangle%}
1063 \\\selectfont
1064 \\\fontencoding\langle#3\rangle%}
1065 \fi
```
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 \ldf@quit to set the main language, restore the category code of the @-sign and call \endinput.

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

Finally we check \originalTeX.

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

Finally we check \originalTeX.
\ldef\ldf@quit{#1} % \ldf@quit This macro interrupts the processing of a language definition file.
\edef\ldf@quit{#1} % TODO. Merge into the next macro? Unused elsewhere
\let\ldf@quit\relax % \ldf@finish This macro takes one argument. It is the name of the language that was defined in the language definition file.
\loadlocalcfg{#1} % 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.
\bbl@beforeldf{% % 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.
\main@language % 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.
\bbl@id@assign % We also have to make sure that some code gets executed at the beginning of the document, either when the aux file is read or, if it does not exist, when the \AtBeginDocument is executed. Languages do not set \pagedir, so we set here for the whole document to the main \bodydir.
\def\bbl@beforestart{% % \bbl@warning{\ldef\ldf@quit{#1} %} % \bbl@warning{\ldef\ldf@quit{#1} %} %
A bit of optimization. Select in heads/foots the language only if necessary.

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 \TeX{} is used). It is used only at one place, namely when \initiate@active@char is called (which is ignored if the char has been made active before). Because \sanitize can be undefined, we put the definition inside a conditional.

Items are added to the lists without checking its existence or the original catcode. It does not hurt, but should be fixed. It's already done with \fss@catcodes, added in 3.10.

\bbl@trace{Shorhands}

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

\bbl@remove@special The companion of the former macro is \bbl@remove@special. It removes a character from the set macros \dospecials and \sanitize, but it is not used at all in the babel core.
\def\bbl@remove@special#1{\begingroup\def\x##1##2{\ifnum`#1=`##2\noexpand\@empty\else\noexpand##1\noexpand##2\fi}%\def\do{\x\do}\def\@makeother{\x\@makeother}\edef\x{\endgroup\def\noexpand\dospecials{\dospecials}\expandafter\ifx\csname @sanitize\endcsname\relax\else\def\noexpand\@sanitize{\@sanitize}\fi}\x}\initiate@active@char

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

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

However, a deactivated shorthand (with \texttt{\bbl@deactivate} is defined as \texttt{\activprefix\ "\normalchar\"}.

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

\def\bbl@active@def#1#2#3#4{% @namedef{#3#1}{\expandafter\ifx\csname#2@sh@#1\endcsname\relax\bbl@afterelse\csname#4#1\endcsname\string#1\else\bbl@afterfi\csname#2@sh@#1\string#1\endcsname\fi}}%\initiate@active@char

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.

\def\bbl@active@def#1#2#3#4{% @namedef{#3#1}{\expandafter\ifx\csname#2@sh@#1\endcsname\relax\bbl@afterelse\csname#4#1\endcsname\string#1\else\bbl@afterfi\csname#2@sh@#1\string#1\endcsname\fi}}%\initiate@active@char calls \initiate@active@char with 3 arguments. All of them are the same character with different catcodes: active, other (\string'ed) and the original one. This trick simplifies the code a lot.

\def\bbl@active@char#1{% \bbl@ifunset{\activchar\string#1}\{\bbl@withactive\begin{macro}{\@initiate@active@char}\expandafter\@initiate@active@char\end{macro}\}\end{macro}\}}%\initiate@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).

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

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

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

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 shortands in its first argument and a shorthand character in its second argument. It will expand to either `\bbl@firstcs` or `\bbl@scndcs`. Hence two more arguments need to follow it.

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

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

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

In some circumstances it is necessary to be able to reset the shorthand to its 'normal' value (usually the character with catcode 'other') on the fly. For this purpose the switch `\@safe@actives` is available. The setting of this switch should be checked in the first level expansion of `\active@char` (char).

When this expansion mode is active (with `\@safe@actives`), something like "13":13 becomes "12";12 in an `\edef` (in other words, shortands are 'string'ed). This contrasts with `\protected@edef`, where catcodes are always left unchanged. Once converted, they can be used safely even after this expansion mode is deactivated (with `\@safe@activesfalse`).

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

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

\begin{verbatim}
def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}
\end{verbatim}
\bbl@activate 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.
\bbl@deactivate

1348 \chardef \bbl@activated \z@ \\
1349 \def \bbl@activate #1{% \\
1350 \chardef \bbl@activated \@ne \\
1351 \bbl@withactive {\expandafter \let \expandafter} #1% \\
1352 \csname bbl@active@\string #1\endcsname \\
1353 \def \bbl@activate #1{% \\
1354 \chardef \bbl@activated \tw@ \\
1355 \bbl@withactive {\expandafter \let \expandafter} #1% \\
1356 \csname bbl@normal@\string #1\endcsname

\bbl@firstcs These macros are used only as a trick when declaring shorthands.
\bbl@scndcs

1357 \def \bbl@firstcs #1 #2 {\csname #1\endcsname} \\
1358 \def \bbl@scndcs #1 #2 {\csname #2\endcsname}

\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.

1359 \def \babel@texpdf #1 #2 #3 #4 {% \\
1360 \ifx \textorpdfstring \@undefined \\
1361 \textormath {#1}{#3} \\
1362 \else \\
1363 \textorpdfstring {\textormath {#1}[#3][#2]} {#2} \\
1364 \% \textorpdfstring {\textormath {#1}[#3][#2][#4]} {#4} \\
1365 \fi \\
1366 \} \\
1367 \def \declare@shorthand #1 #2 #3 #4 {% \\
1368 \ifx \empty \@decl@short {#1} #2 #3 #4 \\
1369 \else \\
1370 \expandafter \let \csname #1@sh@\string #2@sel\endcsname \bbl@scndcs \\
1371 \bbl@ifunset {#1@sh@\string #2@} {} \\
1372 \expandafter \let \csname #1@sh@\string #3@\endcsname \bbl@tempa \\
1373 \bbl@ifunset {#1@sh@\string #2@\string #3@} {} \\
1374 \expandafter \let \csname #1@sh@\string #3@\endcsname \bbl@tempa \\
1375 \else \\
1376 \bbl@info \\
1377 \{\textbf{Redefining #1 shorthand \string #2\% in language \CurrentOption}\} \\
1378 \} \\
1379 \else \\
1380 \@namedef {#1@sh@\string #2@} {#3} \\
1381 \else \\
1382 \expandafter \let \csname #1@sh@\string #2@\endcsname \bbl@firstcs \\
1383 \bbl@ifunset {#1@sh@\string #2@\string #3@} {} \\
1384 \{\textbf{Redefining #1 shorthand \string #2\% in language \CurrentOption}\} \\
1385 \else \\
1386 \bbl@info \\
1387 \{\textbf{Redefining #1 shorthand \string #2\% in language \CurrentOption}\} \\
1388 \@namedef {#1@sh@\string #2@\string #3@} {#4} \\
1389 \} \\
1390 \fi \\
1391 \fi
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
\def\textormath{% 
  \ifmmode
    \expandafter\@secondoftwo
  \else
    \expandafter\@firstoftwo
  \fi}

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

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

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

\textormath
\def\useshorthands{% 
  \@ifstar\bbl@usesh@s{% 
    \bbl@usesh@x{}}
\def\bbl@usesh@s#1{% 
  \bbl@usesh@x
  \bbl@ifshorthand{#1}{
    \def\user@group{user}
    \initiate@active@char{#1}
    #1
  }{
    \bbl@error{shorthand-is-off}{#1}
  }}
\def\bbl@usesh@x#1#2{% 
  \bbl@ifshorthand{#2}{
    \def\user@group{user}
    \initiate@active@char{#2}
    #1
  }{
    \bbl@error{shorthand-is-off}{#2}
  }}

\textormath
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.

\textormath
\def\user@language@group{user@\language@group}
\def\bbl@set@user@generic#1#2{% 
  \bbl@ifunset{user@generic@active#1}{
    \bbl@active@def#1\user@language@group{user@active}{user@generic@active}
    \bbl@active@def#1\user@group{user@generic@active}{language@active}
    \expandafter\edef\csname#2@sh@#1@@\endcsname{\expandafter\noexpand\csname normal@char#1\endcsname}
    \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{\expandafter\noexpand\csname user@active#1\endcsname}
  }{\@empty}
\newcommand\defineshorthand[3][user]{% 
  \edef\bbl@tempa{\zap@space#1 \@empty}
  \bbl@for\bbl@tempb{\bbl@tempa}{% 
    \if\expandafter\@car\bbl@tempb\@nil
      \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}{% 
        \edef\bbl@tempb{\expandafter\string\@car\bbl@tempb\@nil}{% 
          \expandafter\bbl@set@user@generic{\expandafter\string\@car\bbl@tempb\@nil}{\bbl@tempb}{% 
            \fi
            \bbl@error{shorthand-is-off}{#2}{#3}}

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

\textormath
\def\languageshorthands#1{% 
  \edef\language@group{#1}

35
\aliasshorthand

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

\def\aliasshorthand#1#2{% 
  \bbl@ifshorthand{#2}% 
  {\expandafter{\ifx\csname active@char\string#2\endcsname\relax 
    \document@notprerr \notshorthand{#2}% 
    \else \initiate@active@char{#2}% 
    \bbl@ccarg\let{active@char\string#2}{active@char\string#1}% 
    \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}% 
    \bbl@activate{#2}% 
    \fi} 
  \fi}% 
  {\bbl@error{shorthand-is-off}{#1}{}{}}}

@notshorthand

\def@notshorthand#1{\bbl@error{not-a-shorthand}{#1}{}{}}

\shorthandon

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

\newcommand*{\shorthandon}[1]{\bbl@switch@sh\@ne#1\@nnil}
\DeclareRobustCommand*{\shorthandoff}{\ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
\def\bbl@shorthandoff#1#2{\bbl@switch@sh#1#2\@nnil}

\bbl@switch@sh

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

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

36
Note the value is that at the expansion time; eg, in the preamble shorthands are usually deactivated.

\def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
\def\bbl@putsh#1{\bbl@ifunset{bbl@active\string#1}{\bbl@putsh@i#1\@empty\@nnil}{\csname bbl@active\string#1\endcsname}}
\def\bbl@putsh@i#1#2\@nnil{\csname\language@group @sh\string#1\ifx\@empty#2\else\string#2\fi\endcsname}
\ifx\bbl@opt@shorthands\@nnil\else\let\bbl@s@initiate@active@char\initiate@active@char\def\initiate@active@char#1{\bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}\let\bbl@s@switch@sh\bbl@switch@sh\def\bbl@switch@sh#1#2\ifx#2\@nnil\else\bbl@afterfi\bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}\fi\fi
You may want to test if a character is a shorthand. Note it does not test whether the shorthand is on or off.
\newcommand\ifbabelshorthand[3]{\bbl@ifunset{bbl@active\string#1}{#3}{#2}}\bbl@prim@s\bbl@pr@m@s
One of the internal macros that are involved in substituting \prime for each right quote in math mode is \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.
\def\bbl@prim@s{\prime\futurelet\@let@token\bbl@pr@m@s}
\def\bbl@if@primes#1#2{\ifx#1\@let@token\expandafter\@firstoftwo\else\ifx#2\@let@token\bbl@afterelse\expandafter\@firstoftwo\else\bbl@afterfi\expandafter\@secondoftwo\fi\fi}
\begingroup\catcode`^=7 \catcode`*=12 \lccode`^=`\*
\gdef\bbl@pr@m@s{\bbl@if@primes"\pr@@@s{\bbl@if@primes*\pr@@@t\egroup}}\endgroup

\begin{group}
\catcode`\^=7 \catcode`\*=12 \lccode`\*=`\^\lowercase{\gdef\bbl@prim@s{\prime\futurelet\@let@token\bbl@pr@m@s}}\endgroup

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

\OT1dqpos \T1dqpos

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

4.6 Language attributes

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

The macro \languageattribute checks whether its arguments are valid and then activates the selected language attribute. First check whether the language is known, and then process each attribute in the list.

\languageattribute

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

When we end up here the attribute is not selected before. So, we add it to the list of selected attributes and execute the associated \TeX-code.

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 \ extrast... for the current language is extended, otherwise the attribute will not work as its code is removed from memory at \begin{document}.

\bbl@declare@tribute
\bbl@ifattributeset This internal macro has 4 arguments. It can be used to interpret \TeX{} code based on whether a certain attribute was set. This command should appear inside the argument to \texttt{\AtBeginDocument} because the attributes are set in the document preamble, after \texttt{babel} is loaded. The first argument is the language, the second argument the attribute being checked, and the third and fourth arguments are the true and false clauses.

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

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

4.7 Support for saving macro definitions

To save the meaning of control sequences using \texttt{\global\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.
The initialization of a new save cycle: reset the counter to zero.

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

The macro \texttt{babel@save\langle csname\rangle} saves the current meaning of the control sequence \texttt{\langle csname\rangle} 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\langle variable\rangle} saves the value of the variable. \texttt{\langle variable\rangle} can be anything allowed after the \texttt{\the} primitive. To avoid messing saved definitions up, they are saved only the very first time.

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

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

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

4.9 Hyphens

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

\def\bbl@allowhyphens{% \nobreak \hskip 0pt plus 0pt\textdagger.
\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{% \@if unset{bbl@hy@#1#2\@empty}{\csname bbl@#1usehyphen\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#1\fi
\nobreak\hskip\z@skip}
\def\bbl@@usehyphen#1{% \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

The following macro inserts the hyphen char.

\def\bbl@hyphenchar{% \@ifstar{\bbl@hyphen@i@}{\bbl@hyphen@i@\@empty}}
\def\bbl@hy@soft{% \bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
\def\bbl@hy@@soft{% \bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
\def\bbl@hy@hard{% \bbl@usehyphen\bbl@hyphenchar}
\def\bbl@hy@@hard{% \bbl@@usehyphen\bbl@hyphenchar}
\def\bbl@hy@nobreak{% \bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
\def\bbl@hy@@nobreak{% \mbox{\bbl@hyphenchar}}
\def\bbl@hy@repeat{% \bbl@usehyphen{\discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
\def\bbl@hy@@repeat{% \bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
\def\bbl@hy@empty{% \hskip\z@skip}
\def\bbl@hy@@empty{% \discretionary{}{}{}}

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. A space in \bbl@hy@nobreak is redundant. A space in \bbl@hy@nobreak is redundant. A space in \bbl@hy@nobreak is redundant. A space in \bbl@hy@nobreak is redundant.

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

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

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

\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.
4.10 Multiencoding strings

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

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

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

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

```
\DeclareOption{nocase}{}
\let\bbl@opt@strings\@nnil % accept strings=value
\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.

```
\@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

\let\bbl@provstring##1##2{\providecommand##1{##2}}\def\bbl@provstring#1#2{% 
\providecommand#1{#2}% 
\def\bbl@tocglobal#1{\global\let#1\@empty}
\def\bbl@provstring#1#2{% 
\providecommand#1{#2}% 
\bbl@provstring#1#2}% 
\let\bbl@scafter\@empty
\let\bbl@screset\@nnil % local flag - disable 1st stopcommands
\let\StartBabelCommands\bbl@startcmds
\let\bbl@Language\CurrentOption
\fi
\begingroup
\ifx\bbl@opt@strings\@nnil
 \let\bbl@opt@strings\BabelStringsDefault
\fi
\bbl@startcmds@i
\def\bbl@startcmds@i#1#2{\edef\bbl@L{\zap@space#1 \@empty}}
```

43
Parsetheencodinginfotogetthelabel,input,andfontparts.
Selectthebehaviorof\SetString. Therearetwomaincases,dependingofifthereisanoptional
argument: withoutitandstrings=encoded,stringsaredefinedalways;otherwise,theyaresetonly
iftheyarestillundefined (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 isn't in strings) do nothing.
We presume the current block is not loaded, and therefore set (above) a couple of default values to
gobble the arguments. Then, these macros are redefined if necessary according to several
parameters.
There are two versions of $\texttt{\bbl@scswitch}$. The first version is used when \texttt{ldfs} are read, and it makes sure $\langle\text{group}\rangle\langle\text{language}\rangle$ is reset, but only once ($\texttt{\bbl@screset}$ is used to keep track of this). The second version is used in the preamble and packages loaded after \texttt{babel} and does nothing. The macro $\texttt{\bbl@forlang}$ loops $\texttt{\bbl@L}$, but its body is executed only if the value is in $\langle\text{group}\rangle\langle\text{language}\rangle$ (used in \texttt{ldfs}), or \texttt{\date} is defined (after \texttt{babel} has been loaded). There are also two versions of $\texttt{\bbl@forlang}$. The first one skips the current iteration if the language is not in $\langle\text{group}\rangle\langle\text{language}\rangle$ (used in \texttt{ldfs}), and the second one skips undefined languages (after \texttt{babel} has been loaded).

Now we define commands to be used inside $\langle\text{group}\rangle\langle\text{language}\rangle$.

**Strings**

The following macro is the actual definition of $\texttt{\SetString}$ when it is “active”. First save the “switcher”. Create it if undefined. Strings are defined only if undefined (i.e., like $\texttt{\providescommand}$). With the event $\texttt{stringprocess}$ you can preprocess the string by manipulating the value of $\langle\text{group}\rangle\langle\text{language}\rangle$. If there are several hooks assigned to this event, preprocessing is done in the same order as defined. Finally, the string is set.

A little auxiliary command sets the string. TODO: Formerly used with casing. Very likely no longer necessary, although it's used in $\langle\text{group}\rangle\langle\text{language}\rangle$.
Define \SetStringLoop, which is actually set inside \StartBabelCommands. The current definition is somewhat complicated because we need a count, but \count@ is not under our control (remember \SetString may call hooks). Instead of defining a dedicated count, we just “pre-expand” its value.

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

Delaying code  Now the definition of \AfterBabelCommands when it is activated.
\def\bbl@aftercmds#1{
\toks@\expandafter{\bbl@scafter#1}%
\xdef\bbl@scafter{\the\toks@}}

Case mapping  The command \SetCase is deprecated. Currently it consists in a definition with a hack just for backward compatibility in the macro mapping.
\newcommand\SetCase[3][]{\def\bbl@tempa####1####2{\ifx####1\@empty\else
\bbl@carg\bbl@add{extras\CurrentOption}{\bbl@carg\babel@save{c__text_uppercase\string####1_tl}\
\bbl@carg\def{c__text_uppercase\string####1_tl}{####2}\
\bbl@carg\babel@save{c__text_lowercase\string####2_tl}\
\bbl@carg\def{c__text_lowercase\string####2_tl}{####1}\\}
\expandafter\bbl@tempa\fi}
\bbl@tempa##1\@empty\@empty
\bbl@carg\bbl@toglobal{extras\CurrentOption}}}

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.
\newcommand\SetHyphenMap[1]{\bbl@forlang\bbl@tempa{\expandafter\bbl@stringdef\csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}
The following package options control the behavior of hyphenation mapping.

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

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

\setlowbox

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

\setlowbox\trace{Macros related to glyphs}
\def\setlowbox#1\setbox\z@\hbox{\textquotedblright}
\setbox\z@\lower\advance\ht\z@-\ht\tw@ 
\kern-.04em\bbl@allowhyphens

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

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

\ProvideTextCommand{\quotdblbase}{OT1}{}
\ProvideTextCommandDefault{\quotdblbase}{\UseTextSymbol{OT1}{\quotdblbase}}

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

\ProvideTextCommand{\quotesinglebase}{OT1}{}
\UseTextSymbol{OT1}{\quotesinglebase}

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

```
2019 \ProvideTextCommandDefault\{\quotesinglebase\}\{% 
2020 \UseTextSymbol\{OT1\}\{\quotesinglebase\}
\}

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

2021 \ProvideTextCommand\{\guillemetleft\}\{OT1\}\{% 
2022 \ifmmode 
2023 \ll 
2024 \else 
2025 \save@sf@q\{\nobreak 
2026 \raise.2ex\hbox\{$\scriptscriptstyle\ll$}\bb@allowhyphens\%
2027 \fi\} 
2028 \ProvideTextCommand\{\guillemetright\}\{OT1\}\{% 
2029 \ifmmode 
2030 \gg 
2031 \else 
2032 \save@sf@q\{\nobreak 
2033 \raise.2ex\hbox\{$\scriptscriptstyle\gg$}\bb@allowhyphens\%
2034 \fi\}
```

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

```
2049 \ProvideTextCommandDefault\{\guillemetleft\}\{% 
2050 \UseTextSymbol\{OT1\}\{\guillemetleft\}\%
2051 \ProvideTextCommandDefault\{\guillemetright\}\{% 
2052 \UseTextSymbol\{OT1\}\{\guillemetright\}\%
2053 \ProvideTextCommandDefault\{\guillemotleft\}\{% 
2054 \UseTextSymbol\{OT1\}\{\guillemotleft\}\%
2055 \ProvideTextCommandDefault\{\guillemotright\}\{% 
2056 \UseTextSymbol\{OT1\}\{\guillemotright\}\%
```

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

```
2057 \ProvideTextCommand\{\guilsinglleft\}\{OT1\}\{% 
2058 \ifmmode 
2059 <\%
2060 \else 
2061 \save@sf@q\{\nobreak 
2062 \raise.2ex\hbox\{$\scriptscriptstyle<$}\bb@allowhyphens\%
2063 \fi\}
2064 \ProvideTextCommand\{\guilsinglright\}\{OT1\}\{% 
2065 \ifmmode 
2066 >\%
2067 \else 
2068 \save@sf@q\{\nobreak 
2069 \raise.2ex\hbox\{$\scriptscriptstyle>$}\bb@allowhyphens\%
2070 \fi\}
```

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

```
2071 \ProvideTextCommandDefault\{\guilsinglleft\}\{% 
49
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.

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

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

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

\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, \texttt{(stipcevic@olimp.irb.hr)}.

\def\crrtic@{\hrule height0.1ex width0.3em}
\def\crttic@{\hrule height0.1ex width0.33em}
\def\ddj@{
\setbox0\hbox{d}
\dimen@=\ht0
\advance\dimen@1ex
\dimen@.45\dimen@
\dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
\advance\dimen@ii.5ex % correction for the dash position
\advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
\dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
\leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
\def\DDJ@{
\setbox0\hbox{D}
\dimen@=.55\ht0
\dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
\advance\dimen@ii.15\fontdimen7\font % correction for the dash position
\advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
\dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
\leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}

\DeclareTextCommand{\dj}{OT1}{\ddj@ d}
\DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

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

\ProvideTextCommandDefault{\dj}{\UseTextSymbol{OT1}{\dj}}
\ProvideTextCommandDefault{\DJ}{\UseTextSymbol{OT1}{\DJ}}

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

\DeclareTextCommand{\S}{OT1}{SS}
\ProvideTextCommandDefault{\S}{\UseTextSymbol{OT1}{\S}}

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 mathmode. They are defined with \ProvideTextCommandDefault, but this is very likely not required because their definitions are based on encoding-dependent macros.
The ‘german’ single quotes.

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

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

The ‘german’ double quotes.

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

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

The ‘french’ single guillemets.

\ProvideTextCommandDefault{\flq}{\textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
\ProvideTextCommandDefault{\frq}{\textormath{\guilsinglright}{\mbox{\guilsinglright}}}

The ‘french’ double guillemets.

\ProvideTextCommandDefault{\flqq}{\textormath{\guillemetleft}{\mbox{\guillemetleft}}}
\ProvideTextCommandDefault{\frqq}{\textormath{\guillemetright}{\mbox{\guillemetright}}}

4.12.4 Umlauts and tremas

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

\umlauthigh
\umlautlow

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

\def\umlauthigh{\def\bbl@umlauta##1{\leavevmode\bgroup\accent\csname f@encoding dqpos\endcsname##1\bbl@allowhyphens\egroup}}
\def\umlautlow{\def\bbl@umlauta{\protect\lower@umlaut}}
\def\umlauta{\bbllauta@\bbl@umlauta\umlauta@\umlauta@}}
\def\umlautlow{\def\bbl@umlauta{\protect\lower@umlaut}}
\def\umlautlow{\def\bbl@umlauta{\protect\lower@umlaut}}
The command \lower@umlaut is used to position the " closer to the letter. We want the umlaut character lowered, nearer to the letter. To do this we need an extra \texttt{dimen} register.

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 \texttt{.45ex} depends on the METAFONT parameters with which the fonts were built. (Just try out, which value will look best.) If the new x-height is too low, it is not changed. Finally we call the \texttt{accent} primitive, reset the old x-height and insert the base character in the argument.

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

For all vowels we declare " to be a composite command which uses \texttt{bb@umlaut} or \texttt{bb@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 \texttt{bb@umlaut} and/or \texttt{bb@umlaute} for a language in the corresponding \texttt{ldf} (using the babel switching mechanism, of course).

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

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

\begin{verbatim}
\ifx\lenglish\@undefined\chardef\lenglish\z@\fi
\ifx\lunhyphenated\@undefined\newlanguage\lunhyphenated\fi
\end{verbatim}

\subsection{Layout}

Layout is mainly intended to set bidi documents, but there is at least a tool useful in general.
4.14 Load engine specific macros

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

4.15 Creating and modifying languages

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

53
\let\bbl@extend@ini@gobble
\newcommand\babelprovide[2][]{%  
\let\bbl@savelangname\languagename  
\edef\bbl@savelocaleid{\the\localeid}%  
% Set name and locale id  
\edef\languagename{#2}%  
\bbl@id\assign  
% Initialize keys  
\bbl@vforeach{captions,date,import,main,script,language,%  
  hyphenrules,linebreaking,justification,mapfont,maparabic,%  
  mapdigits,intraspace,intrapenalty,onchar,transforms,alph,%  
  Alph,labels,labels*,calendar,date,casing,interchar}%  
% Initialize keys  
\bbl@csarg\let{KVP@##1}\@nnil}-%  
\global\let\bbl@release@transforms\@empty  
\global\let\bbl@release@casing\@empty  
\let\bbl@calendars\@empty  
\global\let\bbl@inidata\@empty  
\global\let\bbl@extend@ini\@gobble  
\global\let\bbl@included@inis\@empty  
\gdef\bbl@key@list{;}-%  
\bbl@forkv{#1}{%  
\in@{/}{##1}% With /, (re)sets a value in the ini  
\ifin@  
\global\let\bbl@extend@ini\bbl@extend@ini@aux  
\bbl@renewinikey##1\@@{##2}-%  
\else  
\bbl@csarg\ifx{KVP@##1}\@nnil\else  
\bbl@error{unknown-provide-key}{##1}{}{}%  
\fi  
\bbl@csarg\def{KVP@##1}{##2}-%  
\fi}-%  
\chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini  
\bbl@ifunset{date#2}\z@{\bbl@ifunset{bbl@llevel@#2}\tw@}-%  
% == init ==  
\ifx\bbl@screset\@undefined\bbl@ldfinit\fi-%  
% == date (as option) ==  
% \ifx\bbl@KVP@date\@nnil\else  
% \fi  
% == import, captions ==  
% \ifx\bbl@KVP@import\@nnil\else  
% \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%  
% {\ifx\bbl@initoload\relax\empty% do setup linebreak, only in 3 cases:  
% \ifcase\bbl@howloaded\let\bbl@bkflag\empty% new  
% \let\bbl@bkflag\empty% new  
% \else\let\bbl@bkflag\empty\let\bbl@bkflag\empty\fi%  
% \fi%  
% \% == import, captions ==  
% \ifx\bbl@KVP@import\@nnil\else  
% \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%  
% {\ifx\bbl@initoload\relax\empty% do setup linebreak, only in 3 cases:  
% \ifcase\bbl@howloaded\let\bbl@bkflag\empty% new  
% \let\bbl@bkflag\empty% new  
% \else\let\bbl@bkflag\empty\let\bbl@bkflag\empty\fi%  
% \fi%  
% \endgroup%  
% \else%  
% \xdef\bbl@KVP@import{\bbl@initoload}%  
% \fi%
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.

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.
\ifset{bbl@chrng}{\languagename}{% 
  \directlua{ 
    Babel.set_chranges_b(\bbl@cl{sbcp}', \bbl@cl{chrng}') }% 
\fi
\ifx\bbl@KVP@onchar@\nil\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@ 
\ifx\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].lg = \the\nameuse{l@\languagename} space
  end
}%
\fi
\bbl@xin@{ fonts }{ \bbl@KVP@onchar space}%
\ifin@ 
\ifset{bbl@lsys}{\languagename}{\bbl@provide@lsys{\languagename}}{}%
\ifset{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
}%
\if\bbl@mapselect@\undefined % TODO. almost the same as mapfont
  \AtBeginDocument{% 
    \bbl@patchfont{{\bbl@mapselect}}%
    {\selectfont}}%
  \def\bbl@mapselect{% 
    \let\bbl@mapselect=relax
    \edef\bbl@prefontid{\fontid\font}%
  }% 
  \def\bbl@mapdir#1{% 
    \begingroup\% Force text mode
  \def\language{#1}%
  \let\bbl@ifrestoring=\firstoftwo % To avoid font warning
  \bbl@switchfont
  \edef\fontid\font@2 % A hack, for the pgf nullfont hack
  \directlua{
\bbl@exp{\\bbl@add\\bbl@mapselect{\\bbl@mapdir{\languagename}}}%
\fi
% TODO - catch non-valid values
\fi
% == mapfont ==
% For bidi texts, to switch the font based on direction
\ifx\bbl@KVP@mapfont\@nnil\else
\bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
{\bbl@error{unknown-mapfont}{}}%
\bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
\bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
\ifx\bbl@mapselect\@undefined % TODO. See onchar.
\AtBeginDocument{%
\bbl@patchfont{{\bbl@mapselect}}%
{\selectfont}}%
\def\bbl@mapselect{%
\let\bbl@mapselect\relax
\edef\bbl@prefontid{\fontid\font}}%
\def\bbl@mapdir{}%
{\def\languagename{#1}%
{\def\languagename{##1}%
{\let\bbl@ifrestoring\@firstoftwo % avoid font warning
\bbl@switchfont
directlua{Babel.fontmap
[[\the\localeid]\[\bbl@prefontid\]=\fontid\font}]}%}
\fi
\bbl@exp{\\bbl@add\\bbl@mapselect{\\bbl@mapdir{\languagename}}}%
\fi
% == Line breaking: intraspace, intrapenalty ==
% For CJK, East Asian, Southeast Asian, if interspace in ini
\ifcase\bbl@engine\or
\bbl@xin@{/c}{/c}
\bbl@ifunset{bbl@quote@\languagename}{}%
{\directlua{
Babel.locale_props{\the\localeid}.cjk_quotes = {}
\local cs = 'op'
for c in string.utfvalues(%
[\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
\bbl@provide@intraspase
% == Line breaking: CJK quotes == TODO -> @extras
\ifcase\bbl@engine\or
\bbl@xin@{,/}{c}{/c}
\bbl@ifin@%
\bbl@ifunset{bbl@quote@\languagename}{}%
{\directlua{%
Babel.locale_props{\the\localeid}.cjk_quotes = {}
\local cs = 'op'
for c in string.utfvalues(%
[\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
% == 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,%
% == Casing ==
\@release@casing
\ifx\KVP@casing\@nnil\else
\csarg\def{casing@language}{\@nameuse{bbl@casing@language}\maybextx\KVP@casing}\fi
% == Calendars ==
\ifx\KVP@calendar\@nnil\else
\edef\KVP@calendar{\cl{calpr}}\fi
\def\tempe##1 ##2\@@{% Get first calendar
\def\tempa{##1}}\expandafter\tempe\KVP@calendar\space\@@
\def\tempe##1.##2.##3\@@{%
\def\tempc{##1}%
\def\tempb{##2}}\expandafter\tempe\tempa.\@@
\edef{calpr@language}{\ifx\tempc\@empty\else calendar=\tempc\fi\ifx\tempb\@empty\else ,variant=\tempb\fi}
% == engine specific extensions ==
% Defined in XXXbabel.def
\provide@extra{#2}
% == require.babel in ini ==
% To load or reaload the babel-*.tex, if require.babel in ini
\if\beforestart\relax\else % But not in doc aux or body
\if\rqtex@language\@empty\else\let\BabelBeforeIni\@gobbletwo\fi\fi
\catcode\@=11\relax
\def\CurrentOption{#2}\bbl@input@texini{rqtex@language}\catcode\@=\atcatcode
\let\atcatcode\relax
\global\let{rqtex@language}\relax
\bbl@foreach{rqtex@language}{%
% == transforms ==
\def\CurrentOption{#2}\
\@nameuse{bbl@icsave@#2}\
% == main ==
\ifx\bbl@KVP@main@\@nnil % Restore only if not 'main'
\let\languagename\bbl@savelangname
\chardef\localeid\bbl@savelocaleid\relax
\fi
% ==hyphenrules (apply if current) ==
\ifx\bbl@KVP@hyphenrules@\@nnil\else
\ifnum\bbl@savelocaleid=\localeid
\language\@nameuse{l@\languagename}\
\fi\fi
Depending on whether or not the language exists (based on \date<language>), we define two macros. Remember \bbl@startcommands opens a group.
\def\bbl@provide@new#1{\
\@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
\@namedef{extras#1}{}\
\@namedef{noextras#1}{}\
\bbl@startcommands*{#1}{captions}\
\ifx\bbl@KVP@captions@\@nnil % and also if import, implicit
\def\bbl@tempb##1{% elt for \bbl@captionslist
\ifx##1\@nnil\else
\bbl@exp{\\SetString\##1{\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}\
\expandafter\bbl@tempb
\fi\fi\expandafter\bbl@tempb\bbl@captionslist\@nnil
\else
\ifx\bbl@initoload@relax
\bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
\else
\bbl@read@ini{\bbl@initoload}2% % Same
\fi\fi\StartBabelCommands*{#1}{date}\
\ifx\bbl@KVP@date@\@nnil
\bbl@exp{\\\SetString\\#1{%\bbl@nocaption{\bbl@strip#1}{\#1\bbl@strip#1}}}\
\expandafter\bbl@tempb
\else\fi\StartBabelCommands*{#1}{endcommands}\
\bbl@load@basic{#1}\
% == hyphenmins == (only if new)
\bbl@exp{\gdef<#1hyphenmins>{\{\bbl@ifunset{bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}\
{\bbl@ifunset{bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}}\
% == hyphenrules (also in renew) ==
\bbl@provide@renew#1{\
\ifx\bbl@KVP@captions@\@nnil\else
\expandafter\bbl@exp@\main@language\expandafter{#1}\
\fi\fi\def\bbl@provide@renew#1{\
\ifx\bbl@KVP@captions@\@nnil\else

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 \provide, with hyphenrules and with import.
The reader of babel-...tex files. We reset temporarily some catcodes.

```
\def\bbl@input@texini#1{%
  \bbl@bsphack
  \bbl@exp{%
    \catcode`\%=14 \catcode`\\=0
    \catcode`\{=1 \catcode`\}=2
    \lowercase{\InputIfFileExists{babel-#1.tex}{}{}}%
    \catcode`\%=	he\catcode`\%
    \catcode`\\=	he\catcode`\%
    \catcode`\{=	he\catcode`\{%
    \catcode`\}=	he\catcode`\}%
  }%
  \bbl@esphack}
```

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

```
\def\bbl@iniline#1\bbl@iniline{%
  \@ifnextchar\[\bbl@inisect{\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}{%
  \def\bbl@inisect\[#1\]#2\@@{\def\bbl@section{#1}}%
  \def\bbl@iniskip#1\@@{}% if starts with ;
  \def\bbl@inistore#1=#2\@@{%
    \bbl@trim@def\bbl@tempa{#1}%
    \bbl@trim\toks@{#2}%
    \bbl@xin{;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
    \ifin@
      \bbl@exp{%
        \g@addto@macro\bbl@inidata{%
          \bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}\fi
    \else
      \bbl@exp{%
        \g@addto@macro\bbl@inidata{%
          \bbl@elt{\.identification.}{\bbl@section.}{\bbl@tempa}}}
    \fi
  }%
}
```

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 \babel\provide it's either 1 or 2.

```latex
\def\bbl@loop@ini{% 
\loop 
\if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop 
\endlinechar\m@ne 
\read\bbl@readstream to \bbl@line 
\endlinechar`^^M 
\if\bbl@line\empty\else 
 \expandafter\bbl@iniline\bbl@line\bbl@iniline 
\fi 
\repeat}
\ifx\bbl@readstream\@undefined 
\csname newread\endcsname\bbl@readstream 
\fi 
\def\bbl@read@ini#1#2{% 
\global\let\bbl@extend@ini\@gobble 
\openin\bbl@readstream=babel-#1.ini 
\ifeof\bbl@readstream 
\bbl@error{no-ini-file}{#1}{}{}% 
\else 
% == Store ini data in \bbl@inidata == 
\catcode`\[=12 \catcode`\]=12 \catcode`\=12 \catcode`\&=12 
\catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12 
\bbl@info{Importing 
\ifcase#2font and identification \or basic \fi 
data for \languagename\% 
from babel-#1.ini. Reported}%
\ifnum#2=\z@ 
\global\let\bbl@inidata\@empty 
\let\bbl@inistore\bbl@inistore@min % Remember it's local 
\fi 
\def\bbl@section{identification} 
\bbl@exp{\bbl@inistore tag.ini=#1\@@} 
\bbl@inistore load.level=#2\@@ 
\bbl@loop@ini 
% == Process stored data == 
\bbl@csarg\xdef{lini@\languagename}{#1} 
\bbl@read@ini@aux 
% == 'Export' data == 
\bbl@ini@exports{#2} 
\global\let\bbl@inidata@\empty 
\let\bbl@inistore@\empty 
\let\bbl@inistore@min \% Remember it's local 
\def\bbl@section{identification} 
\bbl@exp{\\bbl@inistore tag.ini=\#1\\@@} 
\bbl@inistore load.level=\#2\\@@ 
\bbl@loop@ini 
% == Store ini data in \bbl@inidata == 
\catcode`\[=12 \catcode`\]=12 \catcode`\=12 \catcode`\&=12 
\catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12 
\bbl@info{Exporting \ifcase#2font and identification \or basic \fi 
data for \languagename\% 
from babel-#1.ini. Reported} 
\ifnum#2=\z@ 
\global\let\bbl@inidata\@empty 
\let\bbl@inidata\@empty 
\bbl@exp{\\bbl@add@list\\bbl@ini@loaded{\languagename}} 
\bbl@toglobal\bbl@ini@loaded 
\fi 
\closein\bbl@readstream 
\def\bbl@read@ini@aux{% 
\let\bbl@savestrings\@empty 
\let\bbl@savetoday\@empty 
\let\bbl@savedate\@empty 
\def\bbl@elt##1##2##3{% 
\def\bbl@section{##1} 
\in@{=date.}{=##1}% Find a better place 
\ifin@ 
 \bbi@ifunset{bbi@inikv@##1}\% 
 \{\bbi@ini@calendar{##1}} 
\fi 
\bbi@ifunset{bbi@inikv@##1}\% 
\{\csname bbl@ini@#2@endcsname{##3}}} 
\bbl@ini@loaded 
\bbl@read@ini@aux 
\fi 
\bbl@ini@loaded 
\bbl@read@ini@aux 
\fi 
\bbl@ini@loaded 
\bbl@read@ini@aux 
\fi 
\bbl@ini@loaded 
\bbl@read@ini@aux 
\fi 
\bbl@ini@loaded 
\bbl@read@ini@aux 
\fi
A variant to be used when the ini file has been already loaded, because it's not the first \babelprovide for this language.

```latex
2841 \def\bbl@extend@ini@aux#1{%
2842 \bbl@startcommands*{#1}{captions}%
2843 % Activate captions/... and modify exports
2844 \bbl@csarg\def\inikv@captions.licr{####1####2}{%
2845 \setlocalecaption{####1}{####2}{%####1####2}%
2846 \def\bbl@ini@captions{####1####2}{%
2847 \bbl@stringdef####1####2{\gdef####1####2{}}%
2848 \def\bbl@exportkey####1####2####3{%
2849 \bbl@ifunset{bbl@@kv@####2}{}%
2850 {\expandafter\ifx\csname bbl@@kv@####2\endcsname\@empty\else
2851 \bbl@exp{\global\let\bbl@####1\languagename\bbl@####2}@{}}%
2852 }%####1####2####3
2853 \fi}%####1####2####3
2854 % As with \bbl@read@ini, but with some changes
2855 \bbl@read@ini@aux
2856 \bbl@ini@exports\tw@
2857 % Update inidata@lang by pretending the ini is read.
2858 \def\bbl@section{####1}%
2859 \bbl@iniline=####3\bbl@iniline}
2860 \csname bbl@inidata@#1\endcsname
2861 \global\bbl@csarg\let\bbl@inidata\bbl@inidata
2862 % And from the import stuff
2863 \StartBabelCommands*{#1}{date}%
2864 \def\bbl@stringdef####1####2{\gdef####1####2{}}%
2865 \bbl@savetoday
2866 \bbl@savedate
2867 \bbl@endcommands}
```

A somewhat hackish tool to handle calendar sections. TODO. To be improved.

```latex
2868 \def\bbl@ini@calendar#1{%
2869 \lowercase{\def\bbl@tempa{=#1=}}%
2870 \bbl@replace\bbl@tempa{=date.gregorian}{}%
2871 \bbl@replace\bbl@tempa{=date.}{}%
2872 \in@{.licr=}{#1=}%
2873 \ifin@
2874 \ifcase\bbl@engine
2875 \bbl@replace\bbl@tempa{.licr=}{}%
2876 \else
2877 \let\bbl@tempa\relax
2878 \fi
2879 \fi
2880 \ifx\bbl@tempa\relax\else
2881 \bbl@replace\bbl@tempa={\relax}{}
2882 \ifx\bbl@tempa\empty\else
2883 \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2884 \fi
2885 \bbl@exp{%
2886 \def\bbl@inikey##1##2##3{%
2887 \\\bbl@inidate####1...\relax####2\{\bbl@####3}%
2888 \fi}
```

A key with a slash in \babelprovide replaces the value in the ini file (which is ignored altogether). The mechanism is simple (but suboptimal): add the data to the ini one (at this point the ini file has not yet been read), and define a dummy macro. When the ini file is read, just skip the corresponding key and reset the macro (in \bbl@inistore above).

```latex
2889 \def\bbl@renewinikey##1##2##3{%
2890 \edef\bbl@tempa{\zap@space####1####2##3}{%
2891 \edef\bbl@tempb{\zap@space####1####2##3}{%
2892 \bbl@trim\toks@{####3}{%
2893 \bbl@exp{%
2894 \edef\\\\bbl@key@list{\bbl@key@list \bbl@####3}%
2895 \bbl@tempa/\bbl@tempb;}%
```

64
The previous assignments are local, so we need to export them. If the value is empty, we can provide a default value.

The 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 do (as a private use extension). For consistency with other single-letter subtags or ‘singletons’, here is considered an extension, too.

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 do (as a private use extension). For consistency with other single-letter subtags or ‘singletons’, here is considered an extension, too.
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\toks@}
}
```

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

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

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

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

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

```
\def\bbl@inikv@counters#1#2{\% 
  \bbl@ifsamestring{#1}{digits}\%\n  \bbl@exp{\\\g@addto@macro\bbl@release@casing{\\\bbl@casemapping{\bbl@maybextx#1}{\languagename}{\unexpanded{#2}}}}\%
  \ifin@\$casing.\{$#1\% eg, casing. Uv = uV
  \ifin@
    \lowercase{\def\bbl@tempb{#1}}\%
    \bbl@replace\bbl@tempb{casing.}{}\%
    \bbl@exp{\\\g@addto@macro\bbl@release@casing{\\\bbl@casemapping{\bbl@maybextx#1}{\languagename}{\unexpanded{#2}}}}\%
  \else
    \bbl@inikv{#1}{#2}\%
  \fi}
```

66
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 LIFCR, in that order.

```latex
\ifcase\bbl@engine
  \bbl@csarg\def{inikv@captions.licr}#1#2{\bbl@ini@captions@aux{#1}{#2}}
\else
  \def\bbl@ini@captions#1#2{\bbl@ini@captions@aux{#1}{#2}}
\fi
```

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

```latex
\def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
  \bbl@replace\bbl@tempa{.template}{}
  \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}
  \bbl@replace\bbl@toreplace{[ ]}{\csname}
  \bbl@replace\bbl@toreplace{]}{\endcsname{}}
  \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}
  \ifin@
    \@nameuse{bbl@patch\bbl@tempa}
    \global\bbl@csarg\let{\bbl@tempa fmt@\languagename}{\bbl@toreplace}
  \fi

  \bbl@exp{\gdef\fnum@\bbl@tempa{\bl@ifunset{bbl@\bbl@tempa fmt@\languagename}\bbl@toreplace}}
}\ifin@
  \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\languagename}{\bbl@alphnumeral{\bbl@tempc}}
\else
  \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
  \bbl@csarg\protected@xdef{cntr@#1@\languagename}{\bbl@tempb*}
\fi
```

The auxiliary macro for captions define \caption{name}.
**Labels.** Captions must contain just strings, no format at all, so there is new group in ini files.

```latex
\def\bbl@list@the{% part, chapter, section, subsection, subsubsection, paragraph, %
  subparagraph, enumi, enumii, enumiii, enumiv, equation, figure, %
  table, page, footnote, mpfootnote, mpfn}\def\bbl@map@cnt#1{% #1: roman, etc, // #2: enumi, etc
  \bbl@ifunset{bbl@map@#1@\languagename}{\@nameuse{#1}}{%\@nameuse{bbl@map@#1@\languagename}}\def\bbl@inikv@labels#1#2{%\in@{.map}{#1}\ifin@
    \ifx\bbl@KVP@labels\@nnil\else
      \bbl@xin@{ map }{ \bbl@KVP@labels\space}\ifin@
      \def\bbl@tempc{#1}\bbl@replace\bbl@tempc{.map}{}\in@{,#2,}{,arabic, roman, Roman, alph, Alph, fnsymbol,}%
      \bbl@exp{%\gdef\bbl@map@\bbl@tempc @\languagename{\ifin@\#2\else\localecounter{#2}\fi}}\bbl@foreach\bbl@list@the{%\bbl@ifunset{the##1}{}\bbl@exp{\let\bbl@tempd\bbl@sreplace\bbl@tempd%{\bbl@tempc}{##1}\bbl@sreplace\bbl@tempd%{\@empty @\bbl@tempc}<c@##1>}{\bbl@map@cnt{\bbl@tempd}{##1}}}\expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
      \toks@\expandafter\expandafter\expandafter{\csname the##1\endcsname}{\the\toks@}\expandafter\edef\csname the##1\endcsname{{\the\toks@}}\fi}}% 
\else
  % The following code is still under study. You can test it and make
  % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
  % language dependent.
  \in@{enumerate.}{#1}\ifin@
    \def\bbl@tempa{#1}\bbl@replace\bbl@tempa{enumerate.}{}\def\bbl@toreplace{#2}\bbl@replace\bbl@toreplace{[ \]}{\nobreakspace{}}\bbl@replace\bbl@toreplace{[}{\csname the}\
    \bbl@replace\bbl@toreplace{]}{\endcsname{}}\toks@\expandafter\expandafter\expandafter{\csname the##1\endcsname}{\the\toks@}\expandafter\edef\csname the##1\endcsname{{\the\toks@}}\fi}}
\fi %
\fi %
\else
\fi %
\def\bbl@tempa[#1]{\bbl@replace\bbl@tempa{enumerate.}{}}\def\bbl@toreplace[#2]{\bbl@replace\bbl@toreplace{[ \]}{|nobreakspace|}}\bbl@replace\bbl@toreplace{[ }{|csname the|}\bbl@replace\bbl@toreplace{|endcsname|}\bbl@toreplace\expandafter\expandafter\expandafter{\csname the##1\endcsname}{\the\toks@}\expandafter\edef\csname the##1\endcsname{{\the\toks@}}% T000. Execute only once:
\bbl@exp{% \bbl@add\<extras\languagename>{}}
```

68
To show correctly some captions in a few languages, we need to patch some internal macros, because the order is hardcoded. For example, in Japanese the chapter number is surrounded by two string, while in Hungarian is placed after. These replacement works in many classes, but not all. Actually, the following lines are somewhat tentative.

```
\def\bbl@chaptype{chapter}
\ifx\@makechapterhead\@undefined
  \let\bbl@patchchapter\relax
\else\ifx\thechapter\@undefined
  \let\bbl@patchchapter\relax
\else\ifx\ps@headings\@undefined
  \let\bbl@patchchapter\relax
\else
  \def\bbl@patchchapter{\
    \global\let\bbl@patchchapter\relax
    \gdef\bbl@chfmt{\
      \bbl@ifunset{bbl@\bbl@chaptype fmt@\languagename}{}{bl@cl{calpr}},\
      \bbl@cli{convert=convert=}%
    }
    \bbl@add\appendix{\def\bbl@chaptype{appendix}}% Not harmful, I hope
    \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
    \bbl@patchchapter
  }
\fi\fi\fi
\else
  \def\bbl@patchpart{\bbl@patchchapter}
\fi
\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

```
\let\bbl@calendar\@empty
\DeclareRobustCommand{\localedate}[1]{\bbl@localedate{#1}}
\def\bbl@localedate#1#2#3#4{\
  \begingroup
  \edef\bbl@they{#2}\
  \edef\bbl@them{#3}\
  \edef\bbl@thed{#4}\
  \edef\bbl@tempe{\
    \bbl@ifunset{bbl@calpr@\languagename}{}{bbl@calpr},\
    \bbl@cli{convert=convert=}%
  }
  \bbl@replace\bbl@tempe{ }{}\
  \let\bbl@ld@calendar\@empty
  \let\bbl@ld@variant\@empty
```

69
Dates will require some macros for the basic formatting. They may be redefined by language, so "semi-public" names (camel case) are used. Oddly enough, the CLDR places particles like "de" inconsistently in either in the date or in the month name. Note after `\@replace \toks@ contains the resulting string, which is used by `\@replace@finish@iii (this implicit behavior doesn’t seem a good idea, but it’s efficient).
Language and Script values to be used when defining a font or setting the direction are set with the following macros.

**Macro Syntax**

```latex
\bbl@provide@lsys{#1}(%
  \bbl@ifunset{bbl@lname@#1}{}%
  \bbl@loadinfo{#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@addlist{lsys@#1}{Script=\bbl@cs{sname@#1}}%
  \bbl@ifunset{bbl@lname@#1}{}%
  \bbl@csarg\bbl@addlist{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
\ifcase\bbl@engine
or\or
  \bbl@ifunset{bbl@prehc@#1}{}%
  \bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}{}%
    \ifx\bbl@xenohyph\@undefined
      \global\let\bbl@xenohyph\bbl@xenohyph@d
    \else
      \ifx\fontchar\font"200B
    \fi
    \expandafter\selectlanguage\expandafter{\languagename}}%
  \fi
  \bbl@patchfont{\bbl@xenohyph}%
  \expandafter\selectlanguage\expandafter{\languagename}%%
    \expandafter\selectlanguage\expandafter{\languagename}}%
\fi)%
```

**Example**

```latex
\bbl@provide@lsys{#1}(%
  \bbl@ifunset{bbl@lname@#1}{}%
  \bbl@loadinfo{#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@addlist{lsys@#1}{Script=\bbl@cs{sname@#1}}%
  \bbl@ifunset{bbl@lname@#1}{}%
  \bbl@csarg\bbl@addlist{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
\ifcase\bbl@engine
or\or
  \bbl@ifunset{bbl@prehc@#1}{}%
  \bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}{}%
    \ifx\bbl@xenohyph\@undefined
      \global\let\bbl@xenohyph\bbl@xenohyph@d
    \else
      \ifx\fontchar\font"200B
    \fi
    \expandafter\selectlanguage\expandafter{\languagename}}%
  \fi
  \bbl@patchfont{\bbl@xenohyph}%
  \expandafter\selectlanguage\expandafter{\languagename}%%
    \expandafter\selectlanguage\expandafter{\languagename}}%
\fi)%
```
\hyphenchar\font"200B
\else
\bbl@warning
{Neither 0 nor ZERO WIDTH SPACE are available\%
in the current font, and therefore the hyphen\%
will be printed. Try changing the fontspec'\%
'HyphenChar' to another value, but be aware\%
this setting is not safe (see the manual).\%
Reported}%
\hyphenchar\font\defaulthyphenchar
\fi\fi
\fi}%
\fi}

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

A tool to define the macros for native digits from the list provided in the ini file. Somewhat convoluted because there are 10 digits, but only 9 arguments in \TeX. Non-digits characters are kept.
The first macro is the generic "localized" command.
\def\bbl@setdigits#1#2#3#4#5{%\bbl@exp{%\def<\languagename digits>####1{% ie, \langdigits
 \<\bbl@digits@\languagename>####1\@nil}%
 \let<\languagename counter>####1{% ie, \langcounter
 \expandafter<\bbl@counter@\languagename>%
 \csname c@####1\endcsname}%
 \def<\bbl@counter@\languagename>####1{% ie, \bbl@counter@lang
 \expandafter<\bbl@digits@\languagename>\number####1\@nil}}%
 \def\bbl@tempa##1##2##3##4##5{%\bbl@exp{% Wow, quite a lot of hashes! :-(
 \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}}%
 \bbl@tempa}

Alphabetic counters must be converted from a space separated list to an \ifcase structure.
\def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks=\}
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 an special case, for a fixed form (see babel-he.ini, for example).

\newcommand{localenumeral}[2]{\bbl@cs{cntr@#1@languagename}{#2}}
\def{\bbl@localecntr#1#2}{\localenumeral{#2}{#1}}
\newcommand{localecounter}[2]{\expandafter{\bbl@localecntr}{\number{\csname c@#2\endcsname}{#1}}}
\def{\bbl@alphnumeral#1#2}{\expandafter{\bbl@alphnumeral@i}\number#2 76543210\@@#1}
\def{\bbl@alphnumeral@i#1#2#3#4#5#6#7#8}{\ifcase\@car#8\@nil\or % Currently <10000, but prepared for bigger
\bbl@alphnumeral@ii{#9}000000#1\or \bbl@alphnumeral@ii{#9}00000#1#2\or \bbl@alphnumeral@ii{#9}0000#1#2#3\or \bbl@alphnumeral@ii{#9}000#1#2#3#4\else \bbl@alphnum@invalid{>9999}\fi}
\def{\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8}{\bbl@ifunset{bbl@cntr@#1.F.@number#5#6#7#8}{\bbl@cs{cntr@#1.4@languagename}{#5}\bbl@cs{cntr@#1.3@languagename}{#6}\bbl@cs{cntr@#1.2@languagename}{#7}\bbl@cs{cntr@#1.1@languagename}{#8}}{\ifnum#6#7#8>-z@ \bbl@ifunset{bbl@cntr@#1.S.321}{\bbl@cs{cntr@#1.S.321}{#5#6#7#8}}{\bbl@cs{cntr@#1.F.@number#5#6#7#8}}}
\def{\bbl@alphnum@invalid#1}{\bbl@error{alphabetic-too-large}{#1}{}}}
\def{\bbl@localeinfo#1#2}{\bbl@ifunset{bbl@info@#2}{#1}{\bbl@ifunset{bbl@\csname bbl@info@#2\endcsname@languagename}{#1}{\bbl@cs{\csname bbl@info@#2\endcsname@languagename}}}}
\newcommand{localeinfo}[1]{\ifx*#1\@empty % TODO. A bit hackish to make it expandable.
\bbl@afterelse{\bbl@localeinfo{}}{#1}}
\@namedef{bbl@info@name.locale}{lcname}
\@namedef{bbl@info@tag.ini}{lini}
\@namedef{bbl@info@name.english}{elname}
\@namedef{bbl@info@tag.bcp47}{tbcp}
\@namedef{bbl@info@language.tag.bcp47}{lbcp}
\@namedef{bbl@info@tag.opentype}{lotf}

The information in the identification section can be useful, so the following macro just exposes it with a user command.
\def{\bbl@localeinfo@1#2}{\bbl@ifunset{\bbl@info@#2}{\bbl@error{no-ini-info}{}}{\bbl@localeinfo{}}}
\def{\bbl@localeinfo@1}{\ifx*\@empty % TODO. A bit hackish to make it expandable.
\bbl@afterelse{\bbl@localeinfo{}}{}}
\@namedef{bbl@info@script.name}{esname}
\@namedef{bbl@info@script.name.openface}{sname}
\@namedef{bbl@info@script.tag.bcp47}{sbcp}
\@namedef{bbl@info@script.tag.opentype}{sotf}
\@namedef{bbl@info@region.tag.bcp47}{rbcp}
\@namedef{bbl@info@variant.tag.bcp47}{vbcp}
\@namedef{bbl@info@extension.t.tag.bcp47}{extt}
\@namedef{bbl@info@extension.u.tag.bcp47}{extu}
\@namedef{bbl@info@extension.x.tag.bcp47}{extx}

\LaTeX\ needs to know the BCP 47 codes for some features. For that, it expects \texttt{BCPdata} to be defined. While language, region, script, and variant are recognized, extension.⟨s⟩ for singletons may change.

\ifcase\if@engine % Converts utf8 to its code (expandable)
\else % For plain. TODO. It's a quick fix
\renewcommand{\BCPdata}{}
\fi
\fi
\@namedef{bbl@info@casing.tag.bcp47}{casing}
\newcommand\BabelUppercaseMapping[3][]{\DeclareUppercaseMapping\@nameuse{bbl@casing@#1}{#2}{#3}}
\newcommand\BabelTitlecaseMapping[3][]{\DeclareTitlecaseMapping\@nameuse{bbl@casing@#1}{#2}{#3}}
\newcommand\BabelLowercaseMapping[3][]{\DeclareLowercaseMapping\@nameuse{bbl@casing@#1}{#2}{#3}}

The parser for casing and casing.⟨variant⟩.
\def\bbl@casemapping#1#2#3{% 1:variant
    \def\bbl@tempa##1 ##2{% Loop
        \bbl@casemapping@i{##1}%
        \ifx\@empty##2\else\bbl@afterfi\bbl@tempa##2\fi
    }%
    \edef\bbl@templ{\@nameuse{bbl@casing@#2}#1}% Language code
    \def\bbl@tempe{0}% Mode (upper/lower...)
    \def\bbl@tempc{#3 }% Casing list
    \expandafter\bbl@casemapping@ii\bbl@tempc\@empty}
\def\bbl@casemapping@i#1{%
    \def\bbl@tempb{#1}%
    \ifcase\if@engine % Handle utf8 in pdftex, by surrounding chars with {}
    \else
        \@nameuse{regex_replace_all:nnN}{{.}}{{\0}}\bbl@tempb
    \fi
    \expandafter\bbl@casemapping@ii\bbl@tempb\@@}
\def\bbl@casemapping@ii#1#2#3#4#5#6\@@{%
    \in@{#1#3}{<>}% ie, if <u>, <l>, <t>
    \ifin@
        \edef\bbl@tempe{\if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
    \fi
}\@namedef{bbl@info@casing.tag.bcp47}{casing}
\newcommand\BabelUppercaseMapping[3][]{\DeclareUppercaseMapping\@nameuse{bbl@casing@#1}{#2}{#3}}
\newcommand\BabelTitlecaseMapping[3][]{\DeclareTitlecaseMapping\@nameuse{bbl@casing@#1}{#2}{#3}}
\newcommand\BabelLowercaseMapping[3][]{\DeclareLowercaseMapping\@nameuse{bbl@casing@#1}{#2}{#3}}
With version 3.75 \BabelEnsureInfo is executed always, but there is an option to disable it.

\begin{itemize}
\item [\langle\langle∗ More package options \rangle\rangle] ≡ \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\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files \let\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files \let\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files \let\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files \let\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files \let\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files \let\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files
\def\languagename{##1} % \bbl@ensureinfo{##1}}}}\fi \bbl@foreach\bbl@loaded}{\let\bbl@ini@loaded\@empty \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}{\newcommand\getlocaleproperty{\@ifstar\bbl@getproperty@s\bbl@getproperty@x}{\let#1\relax \def\bbl@elt##1##2##3{\bbl@ifsamestring{##1/##2}{#3}{\providecommand#1{##3}{\def\bbl@elt####1####2####3{}}}{}\bbl@cs{inidata@#2}}}{\def\bbl@getproperty@x#1#2#3{\bbl@getproperty@s{#1}{#2}{#3}}}{\ifx#1\relax\bbl@error{unknown-locale-key}{#1}{#2}{#3}{}}{\fi}}{\let\bbl@ini@loaded\@empty \newcommand\LocaleForEach\@ifpackagewith{babel}{ensureinfo=off}{}{\AtEndOfPackage{% Test for plain. \ifx\undefined\bbl@loaded\else\BabelEnsureInfo\fi}}}{\let\bbl@ini@loaded\@empty \newcommand\LocaleForEach}}\newcommand\ShowLocaleProperties#1{\typeout{}\typeout{*** Properties for language '#1' ***}\def\bbl@elt##1##2##3{\typeout{##1/##2 = ##3}}{\@nameuse{bbl@inidata@#1}}{\typeout{*******}}}

5 Adjusting the Babel behavior

A generic high level interface is provided to adjust some global and general settings.
\bbl@ifnum\currentgrouplevel=\z@
  \directlua{Babel.#2}
\expandafter\expandafter\expandafter\@gobble
\fi
}{\bbl@error{adjust-only-vertical}{#1}{}{}}}% Gobbled if everything went ok.
\@namedef{bbl@ADJ@bidi.mirroring@on}{\bbl@adjust@lua{bidi}{mirroring_enabled=true}}
\@namedef{bbl@ADJ@bidi.mirroring@off}{\bbl@adjust@lua{bidi}{mirroring_enabled=false}}
\@namedef{bbl@ADJ@bidi.text@on}{\bbl@adjust@lua{bidi}{bidi_enabled=true}}
\@namedef{bbl@ADJ@bidi.text@off}{\bbl@adjust@lua{bidi}{bidi_enabled=false}}
\@namedef{bbl@ADJ@bidi.math@on}{\let\bbl@noamsmath\@empty}
\@namedef{bbl@ADJ@bidi.math@off}{\let\bbl@noamsmath\relax}
\@namedef{bbl@ADJ@linebreak.sea@on}{\bbl@adjust@lua{linebreak}{sea_enabled=true}}
\@namedef{bbl@ADJ@linebreak.sea@off}{\bbl@adjust@lua{linebreak}{sea_enabled=false}}
\@namedef{bbl@ADJ@linebreak.cjk@on}{\bbl@adjust@lua{linebreak}{cjk_enabled=true}}
\@namedef{bbl@ADJ@linebreak.cjk@off}{\bbl@adjust@lua{linebreak}{cjk_enabled=false}}
\@namedef{bbl@ADJ@justify.arabic@on}{\bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
\@namedef{bbl@ADJ@justify.arabic@off}{\bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
\@namedef{bbl@ADJ@layout.tabular@on}{\ifnum\bbl@tabular@mode=\tw@
  \bbl@adjust@layout{\let@tabular@bbl@NL@@tabular}{\let@tabular@bbl@OL@@tabular}}
\else
  \chardef\bbl@tabular@mode\@ne
\fi}
\@namedef{bbl@ADJ@layout.tabular@off}{\ifnum\bbl@tabular@mode=\tw@
  \bbl@adjust@layout{\let@tabular@bbl@NL@@tabular}{\let@tabular@bbl@OL@@tabular}}
\else
  \chardef\bbl@tabular@mode\z@
\fi}
\@namedef{bbl@ADJ@layout.lists@on}{\bbl@adjust@layout{\let@list@bbl@NL@@list}}
\namedef{bbl@ADJ}@layout.lists@off{\bbl@adjust@layout{\let\list\bbl@OL@list}}
\namedef{bbl@ADJ}@autoload.bcp47@on{\bbl@bcpallowedtrue}
\namedef{bbl@ADJ}@autoload.bcp47@off{\bbl@bcpallowedfalse}
\namedef{bbl@ADJ}@autoload.bcp47.prefix#1{\def\bbl@bcp@prefix{#1}}
\def\bbl@bcp@prefix{bcp47-}
\namedef{bbl@ADJ}@autoload.options#1{\def\bbl@autoload@options{#1}}
\let\bbl@autoload@bcpoptions\@empty
\namedef{bbl@ADJ}@autoload.bcp47.options#1{\def\bbl@autoload@bcpoptions{#1}}
\newif\ifbbl@bcptoname
\namedef{bbl@ADJ}@bcp47.toname@on{\bbl@bcptonametrue\BabelEnsureInfo}
\namedef{bbl@ADJ}@bcp47.toname@off{\bbl@bcptonamefalse}
\namedef{bbl@ADJ}@prehyphenation.disable@nohyphenation{\directlua{ Babel.ignore_pre_char = function(node)
  return (node.lang == \the\csname l@nohyphenation\endcsname)
end }}
\namedef{bbl@ADJ}@prehyphenation.disable@off{\directlua{ Babel.ignore_pre_char = function(node)
  return false
end }}
\namedef{bbl@ADJ}@interchar.disable@nohyphenation{\def\bbl@ignoreinterchar{\ifnum\language=\l@nohyphenation
  \expandafter\@gobble\else\expandafter\@firstofone\fi}}
\namedef{bbl@ADJ}@interchar.disable@off{\let\bbl@ignoreinterchar\@firstofone}
\namedef{bbl@ADJ}@select.write@shift{\let\bbl@restorelastskip\relax
\def\bbl@savelastskip{\let\bbl@restorelastskip\relax
  \ifvmode
    \ifdim\lastskip=\z@
      \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
\def\bbl@savelastskip#1\bbl@restorelastskip{}%}
\namedef{bbl@ADJ}@select.encoding@off{\let\bbl@encoding@select@off\@empty}
\let\bbl@encoding@select\bbl@encoding@select@off
5.1 Cross referencing macros

The \LaTeX{} book states:

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

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

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

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

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

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

\begin{verbatim}
\bbl@trace{Cross referencing macros}
\ifx\bbl@opt@safe\@empty% ie, if 'ref' and/or 'bib'
{\@safe@activestrue
\bbl@ifunset{#1@#2}\relax
{\gdef\@multiplelabels{\@latex@warning@no@line{There were multiply-defined labels}}%
\@latex@warning@no@line{Label `#2' multiply defined}}%
\global\@namedef{#1@#2}{#3}%%
\edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%%
\edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%%
\if\bbl@tempa\bbl@tempb%%
\else%%
\@tempswatrue
\fi}
\end{verbatim}

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

\begin{verbatim}
\CheckCommand*\@testdef[3]{% TODO. With \@samestring?
\@safe@activestrue
\expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname\reserved@a
\else
\@tempswatrue
\fi}
\end{verbatim}

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 \@newl@bel does it. When the label is defined we replace the definition of \bbl@tempa by its meaning. If the label didn't change, \bbl@tempa and \bbl@tempb should be identical macros.

\begin{verbatim}
\def\@testdef#1#2#3{% TODO. With \@samestring?
\@safe@activestrue
\expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname\reserved@a
\else
\@tempswatrue
\fi}
\end{verbatim}
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.

\begin{verbatim}
\let\org@ref\ref
\let\org@pageref\pageref
\ifin@
\edef\bbl@tempc\meaning{\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
\end{verbatim}

\cite 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.

\begin{verbatim}
\let\org@\citex
\ifin@
\bbl@redefine\@citex[#1]{\@safe@activestrue\org@@citex[#1]{\@safe@activesfalse}}
\bbl@redefine\@citex[#1]{\@safe@activestrue\org@@citex[#1]{\@safe@activesfalse}}
\bbl@redefine\@citex[#1]{\@safe@activestrue\org@@citex[#1]{\@safe@activesfalse}}
\else
\bbl@redefinerobust\citex#1{\@safe@activestrue\org@\citex#1{\@safe@activesfalse}}
\bbl@redefinerobust\pageref#1{\@safe@activestrue\org@\pageref#1{\@safe@activesfalse}}
\fi
\end{verbatim}

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.

\begin{verbatim}
\AtBeginDocument{%
@ifpackageloaded{natbib}{%
\def\citex[#1]{\@safe@activestrue\edef\bbl@tempa{\@safe@activesfalse}}
\AtBeginDocument{%
@ifpackageloaded{cite}{%
\def\citex[#1]{\@safe@activestrue\edef\bbl@tempa{\@safe@activesfalse}}
\end{verbatim}

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

\begin{verbatim}
\AtBeginDocument{%
\def\citex[#1]{\@safe@activestrue\edef\bbl@tempa{\@safe@activesfalse}}
\end{verbatim}

80
The macro \nocite which is used to instruct BibTeX to extract uncited references from the database.

\bbl@redefine\nocite#1{\@safe@activestrue\org@nocite{#1}\@safe@activesfalse}

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

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

\def\bbl@bibcite#1#2{\org@bibcite{#1}{\@safe@activesfalse#2}}

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

\def\bbl@cite@choice{\global\let\bibcite\bbl@bibcite\@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}\@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}\global\let\bbl@cite@choice\relax}

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

\AtBeginDocument{\bbl@cite@choice}

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

\def\bbl@bibitem#1{\@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}

\else
\let\org@nocite\nocite\let\org@@citex\@citex\let\org@bibcite\bibcite\let\org@@bibitem\@bibitem\fi

5.2 Marks

\markright Because the output routine is asynchronous, we must pass the current language attribute to the head lines. To achieve this we need to adapt the definition of \markright and \markboth somewhat. However, headlines and footlines can contain text outside marks; for that we must take some actions in the output routine if the 'headfoot' options is used.

We need to make some redefinitions to the output routine to avoid an endless loop and to correctly handle the page number in bidi documents.

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

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

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:

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

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

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

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

\bbl@trace{Preventing clashes with other packages}
\ifx\org@ref\@undefined\else
  \bbl@xin@{R}\bbl@opt@safe
  \ifin@
    \AtBeginDocument{\%}
    \ifpackageloaded@ifthen{%
5.3.2 varioref

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

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

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

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

5.3.3 hhline

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

\AtEndOfPackage{%\@ifpackageloaded{hhline}{-%\expandafter\if\csname normal@char:\endcsname relax
\else
\makeatletter\def\@currname{hhline}\input{hhline.sty}\makeatother
\}}

83
\substitutefontfamily  Deprecated. Use the tools provided by \TeX. The command \substitutefontfamily creates an .fd file on the fly. The first argument is an encoding mnemonic, the second and third arguments are font family names.

\begin{verbatim}
\def\substitutefontfamily#1#2#3{\
  \lowercase{\immediate\openout15=#1#2.fd\relax}\
  \immediate\write15{\
    \string\ProvidesFile{#1#2.fd}\
    [\the\year/\two@digits{\the\month}/\two@digits{\the\day}\
    \space generated font description file]^^J\
    \string\DeclareFontFamily{#1}{#2}{}^^J\
    \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{\relax}{}^^J\
    \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{\relax}{}^^J\
    \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{\relax}{}^^J\
    \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{\relax}{}^^J\
    \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{\relax}{}^^J\
    \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{\relax}{}^^J\
    \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{\relax}{}^^J\
    \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{\relax}{}^^J}\
\closeout15\
}\end{verbatim}

\subsection*{5.4 Encoding and fonts}

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

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

\AtEndOfClass{\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{\if@ifpackageloaded{fontspec} \begin{document} \fontencoding{\latinencoding}\selectfont \def\encodingdefault{\latinencoding} \end{document} \else \begin{document} \fontencoding{T1}\selectfont \def\encodingdefault{T1} \end{document} \fi}

\latintext 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.

\AtBeginDocument{%}
\DeclareTextFontCommand{%}{\selectfont}
\AtEndOfClass{%}
\fi

\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.

\AtBeginDocument{\if@undefined\DeclareTextFontCommand
\DeclareTextFontCommand{%}{\selectfont}
\AtEndOfClass{%}
\fi

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

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

5.5 Basic bidi support

Work in progress. This code is currently placed here for practical reasons. It will be moved to the correct place soon, I hope. It is loosely based on rl\babel.def, but most of it has been developed from scratch. This babel module (by Johannes Braams and Boris Lavva) has served the purpose of typesetting R documents for two decades, and despite its flaws I think it is still a good starting point (some parts have been
copied here almost verbatim), partly thanks to its simplicity. I’ve also looked at ARABI (by Youssef Jabri), which is compatible with babel.

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).

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

```latex
\bbl@trace{Loading basic (internal) bidi support}
\iffode\bbl@engine\else % TODO. Move to txtbabel
  \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200 % Any xe+lua bidi=
    \bbl@error{bidi-only-lua}{}{}{}
  \else % TODO. Move to txtbabel
    \let\bbl@beforeforeign\leavevmode
  \fi
  \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
  \bbl@xebidipar
\def\bbl@loadxebidi#1{\ifx\RTLfootnotetext\@undefined
    \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
    \usepackage#1{bidi}
    \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
    \def\DigitsDotDashInterCharToks{% See the 'bidi' package
      \ifnum\@nameuse{bbl@wdir@\languagename}=\tw@ % 'AL' bidi
        \bbl@digitsdotdash % So ignore in 'R' bidi
      \fi}}
  \fi}
\iff\ifnum\bbl@bidimode>200 % Any xe bidi=
  \iff\expandafter\case\expandafter\@gobble\@gobble\the\bbl@bidimode\or
    \bbl@tentative{bidi=bidi}
  \else
    \bbl@loadxebidi{}
  \fi
  \or
  \bbl@loadxebidi{[rldocument]}
  \or
  \bbl@loadxebidi{}
\fi
\fi
\iff\ifnum\bbl@bidimode=\one % bidi=default
  \let\bbl@beforeforeign\leavevmode
  \iff\odd\bbl@engine % lua
    \directlua{\Babel.attr_dir = \luatexbase.registernumber\'bbl@attr@dir'}
  \else
    \bbl@xebidipar
  \fi
\fi
\fi
\iff\iff\todo? Separate:
  \iff\ifnum\bbl@bidimode=#\one % bidi=default
    \let\bbl@beforeforeign\leavevmode
  \fi
  \iffodd\bbl@engine % lua
    \newattribute{bbl@attr@dir}
  \else
    \bbl@xebidipar
  \fi
\fi
```

86
Now come the macros used to set the direction when a language is switched. First the (mostly)
common macros.

\bbl@trace{Macros to switch the text direction}
\def\bbl@alscripts{% Arabic, Syriac, Thaana,\}%
\def\bbl@rscripts{% Imperial Aramaic, Avestan, Cypriot, Hatran, Hebrew, %
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{%}
\bbl@xin{% \csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
\ifin@
\global\bbl@csarg\chardef{wdir@#1}@ne
\bbl@xin{% \csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
\ifin@
\global\bbl@csarg\chardef{wdir@#1}@tw@
\else
\global\bbl@csarg\chardef{wdir@#1}@z@
\fi
\fi
\ifodd\bbl@engine
\bbl@dirlevel
\chardef\bbl@thetextdir@z@
\chardef\bbl@thepardir@z@
\def\bbl@textdir#1{%}
\ifcase#1
\chardef\bbl@thetextdir@z@
@nameuse{setlatin}%
\bbl@textdir@i\beginL\endL
\else
\chardef\bbl@thetextdir@z@
@nameuse{setnonlatin}%
\bbl@textdir@i\beginR\endR
\fi
\fi\}
\def\bbl@textdir@i#1#2{%}
\ifhmode
\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}
\def\bbl@textdir@i#1#2{%}
\ifcase\bbl@select@type % TODO - strictly, not the right test
\bbl@bodydir[#1]%
\bbl@pardir[#1]% <- Must preceed \bbl@textdir
\fi
\def\bbl@switchdir{%}
\bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys@\languagename}{}%
\bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs@\languagename}{}%
\bbl@exp{[\bbl@setdirs@\bbl@cl@\wdir]}%
\def\bbl@setdirs#1{%}
\bbl@select@type
\bbl@bodydir[#1]%
\bbl@pardir[#1]% <- Must precede \bbl@textdir
\fi
\def\bbl@switchdir{%}
\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
\else
\pdfTeX=0, xetex=2
\newcount\bbl@dirlevel
\chardef\bbl@thetextdir@z@
\chardef\bbl@thepardir@z@
\def\bbl@textdir@i#1{%}
\ifcase\bbl@select@type
\bbl@bodydir[#1]%
\bbl@pardir[#1]% <- Must precede \bbl@textdir
\fi
\def\bbl@textdir@i#1{%}
\bbl@ifcase\bbl@dirlevel
\chardef\bbl@thetextdir@z@
@nameuse{setlatin}%
\bbl@textdir@i\beginL\endL
\else
\chardef\bbl@thetextdir@z@
@nameuse{setnonlatin}%
\bbl@textdir@i\beginR\endR
\fi}
\def\bbl@textdir@i#1{%}
\ifhmode
\bbl@textdir@i\beginL\endL
\else
\bbl@textdir@i\beginR\endR
\fi}
\def\bbl@textdir@i#1{%}
\ifhmode
\bbl@textdir@i\beginL\endL
\else
\bbl@textdir@i\beginR\endR
\fi
\ifodd\bbl@engine
\else
\pdftex=0, xetex=1
\newcount\bbl@dirlevel
\chardef\bbl@thetextdir@z@
\chardef\bbl@thepardir@z@
\def\bbl@textdir@i#1{%}
\ifcase\bbl@select@type
\bbl@bodydir[#1]%
\bbl@pardir[#1]% <- Must precede \bbl@textdir
\fi
\def\bbl@textdir@i#1{%}
The following command is executed only if there is a right-to-left script (once). It activates the \everypar hack for xetex, to properly handle the par direction. Note text and par dirs are decoupled to some extent (although not completely).

\def\bbl@xeeverypar{\ifcase\bbl@thetextdir\else\beginR\fi \let\bbl@severypar\everypar \newtoks\everypar \everypar=\bbl@severypar \ifnum\bbl@bidimode>200 \Any\xe bidi=\let\bbl@textdir\relax\let\bbl@textdir\@empty \AddBabelHook{bidi}{foreign}(%
\def\bbl@tempa{\def\BabelText####1} %
\ifcase\bbl@thetextdir\expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}} %
\else \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}} %
\fi}
\bbl@pardir#1\ifcase#1\relax\setLR\else\setRL\fi %
\ifnum\bbl@bidimode>200 \Any\xe bidi=\fi %
\let\bbl@xeadipar\relax %
\let\bbl@xeeverypar\relax %
\TeXXeTstate\@ne %
\let\bbl@xeeverypar\relax %
\let\bbl@xeadipar\relax %
\ifcase\bbl@thetextdir\else\beginR\fi %
\let\bbl@severypar\everypar %
\newtoks\everypar %
\everypar=\bbl@severypar %
\ifnum\bbl@bidimode>200 \Any\xe bidi=\fi %
A tool for weak L (mainly digits). We also disable warnings with hyperref.
\DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir#1}}
\AtBeginDocument{\ifx\pdfstringdefDisableCommands\@undefined\else %
\begin{document}
\fi %
\let\bbl@textdir\@empty %
\def\bbl@tempa{\def\BabelText####1} %
\ifcase\bbl@thetextdir\expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}} %
\else \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}} %
\fi}
\def\bbl@pardir#1\ifcase#1\relax\setLR\else\setRL\fi %
\fi %
\fi %
\fi %
\fi %
A tool for weak L (mainly digits). We also disable warnings with hyperref.
5.6 Local Language Configuration

\loadlocalcfg

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

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

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 have been processed. The following macro inputs the ldf file and does some additional checks (\input works, too, but possible errors are not caught).

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

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

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

Now define the corresponding loaders. With package options, assume the language exists. With class options, check if the option is a language by checking if the corresponding file exists.

\foreach @language@opts{\
def\tempa{#1}\
\ifx\tempa\opt@main\else\
\ifnum\iniflag<\tw@ % 0 ø (other = ldf)\
\ifunset{ds@#1}\
 DeclareOption[#1]{\loadlanguage[#1]}\fi\fi\fi}
\foreach @classoptionslist{\
def\tempa{#1}\
\ifx\tempa\opt@main\else\
\ifnum\iniflag<\tw@ % 0 ø (other = ldf)\
\ifFileExists{#1.ldf}\
 DeclareOption[#1]{\loadlanguage[#1]}\fi\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 processed before):
\AfterBabelLanguage{#1}\
\ifsamestring\CurrentOption{#1}{\add\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 \provide can't go inside a \DeclareOption; this explains why it's executed directly, with a dummy declaration. Then all languages have been loaded, so we deactivate \AfterBabelLanguage.
6 The kernel of Babel (babel.def, common)

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

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

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

A proxy file for switch.def

\let\bbl@onlyswitch@empty
\input babel.def
\let\bbl@onlyswitch@undefined
</kernel>
\section{Error messages}
They are loaded when \bbl@error is first called. To save space, the main code just identifies them with a tag, and messages are stored in a separate file. Since it can be loaded anywhere, you make sure some catcodes have the right value, although those for |\|, |`|, |^^M|, |%| and |=| are reset before loading the file.

\catcode`\{=1 \catcode`\}=2 \catcode`\#=6 \catcode`:=12 \catcode`\,=12 \catcode`\.=12 \catcode`\(^\)=12 \catcode`\@=11 \catcode`\^=7

\ifx\MessageBreak\@undefined\gdef\bbl@error@i#1#2{\begingroup \newlinechar=`\^^J \def\{\MessageBreak\}{} \errhelp{#2}\errmessage{\#1} \endgroup}\else\gdef\bbl@error@i#1#2{\begingroup \def\{\MessageBreak} \PackageError{babel}{#1}{#2} \endgroup}\fi
\def\bbl@errmessage#1#2#3{\expandafter\gdef\csname bbl@err@#1\endcsname##1##2##3{\bbl@error@i{#2}{#3}}}\bbl@error#1\csname bbl@errmessage\endcsname{not-yet-available}\{Not yet available\} {Find an armchair, sit down and wait} \bbl@errormessage{bad-package-option}\{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.} \bbl@errormessage{base-on-the-fly}\{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.} \bbl@errormessage{undefined-language}\{You haven't defined the language '#1' yet.\} {Perhaps you misspelled it or your installation is not complete\} \bbl@errormessage{shorthand-is-off}\{You can't declare a shorthand turned off (\string#2)\} {Sorry, but you can't use shorthands which have been turned off in the package options\} \bbl@errormessage{not-a-shorthand}\{The character '\string #1' should be made a shorthand character; add the command \string\useshorthands\string(#1\string) to the preamble.\} {I will ignore your instruction\} \bbl@errormessage{not-yet-available} \{Not yet available\} {Find an armchair, sit down and wait}
I can't switch \texttt{#2} on or off--not a shorthand. Maybe you made a typing mistake? I will ignore your instruction.

The attribute \texttt{#2} is unknown for language \texttt{#1}.

Your command will be ignored, type \texttt{<return>} to proceed.

Missing group for string \texttt{\string#1}.

You must assign strings to some category, typically captions or extras, but you set none.

This macro is available only in LuaLaTeX and XeLaTeX.

Consider switching to these engines.

This macro is available only in LuaLaTeX.

Consider switching to that engine.

Unknown key \texttt{#1} in \texttt{\string\babelprovide}.

See the manual for valid keys.

Option \texttt{\bbl@KVP@mapfont} unknown for \texttt{\string#1}.

mapfont. Use 'direction'.

The counter name 'digits' is reserved for mapping.

Use another name.

Currently two-digit years are restricted to the range 0–9999.

There is little you can do. Sorry.

Alphabetic numeral too large (\texttt{#1}).

Currently this is the limit.

I've found no info for the current locale.

The corresponding ini file has not been loaded.

Perhaps it doesn't exist.

See the manual for details.

Unknown field \texttt{#1} in \texttt{\string\BCPdata}.

Perhaps you misspelled it.

See the manual for details.

Unknown key for locale '2':

#3

\string#1 will be set to \texttt{\relax}.

Perhaps you misspelled it.

Currently, \texttt{#1} related features can be adjusted only in the main vertical list.

Maybe things change in the future, but this is what it is.

Currently, layout related features can be adjusted only in vertical mode.

Maybe things change in the future, but this is what it is.
The bidi method ‘basic’ is available only in lualatex. I’ll continue with ‘bidi=default’, so expect wrong results.

{See the manual for further details.}

{Multiple bidi settings inside a group}

{I’ll insert a new group, but expect wrong results.}

{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.}

{Local config file ‘\bbl@opt@config.cfg’ not found}

{Perhaps you misspelled it.}

{Too late for \string\AfterBabelLanguage}

{Languages have been loaded, so I can do nothing}

{Double hyphens aren’t allowed in \string\babelcharclass because it’s potentially ambiguous}

{because it’s potentially ambiguous}

{The allowed range is #1}

{The bidi method ‘basic’ is available only in lualatex. I’ll continue with ‘bidi=default’, so expect wrong results}

{See the manual for further details.}

{Multiple bidi settings inside a group}

{I’ll insert a new group, but expect wrong results.}

{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.}

{Local config file ‘\bbl@opt@config.cfg’ not found}

{Perhaps you misspelled it.}

{Too late for \string\AfterBabelLanguage}

{Languages have been loaded, so I can do nothing}

{Double hyphens aren’t allowed in \string\babelcharclass because it’s potentially ambiguous}

{because it’s potentially ambiguous}

{The allowed range is #1}
4508 \{The 'ldf style doesn't work with \%
4509 \#2,\%
4510 \} but you can use the ini locale instead.\%
4511 Try adding 'provide='' to the option list. You may\%
4512 also want to set 'bidi=' to some value.)\%
4513 \{See the manual for further details.\}
4514 ⟨/errors⟩
4515 ⟨∗ patterns⟩

7 Loading hyphenation patterns

The following code is meant to be read by \input{#2} because it should instruct \LaTeX{} to read hyphenation patterns. To this end the docstrip option patterns is used to include this code in the file hyphen.cfg. Code is written with lower level macros.

\input{#2}
\ProvidesFile{hyphen.cfg}[
\langle\text{date}\rangle\langle\text{version}\rangle Babel hyphens]
\xdef\bbl@format{\jobname}
\def\bbl@version{\langle\text{version}\rangle}
\def\bbl@date{\langle\text{date}\rangle}
\ifx\AtBeginDocument\@undefined
\def\@empty{}\fi
\langle\text{Define core switching macros}\rangle
\process@line

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

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

\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.

\process@synonym\{#1\%
\toks@{}\def\bbl@languages{}

When no languages have been loaded yet, the name following the \texttt{=} will be a synonym for hyphenation register 0. So, it is stored in a token register and executed when the first pattern file has been processed. (The \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.

\process@synonym\{#1\%
\ifnum\last@language=m@ne
\toks@\expandafter{#1}\relax\process@synonym{#1}\%
\else
\process@synonym{#2}\%
\else\process@language{#1#2}{#3}{#4}\%
\fi\ignorespaces}

\process@language

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

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

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

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

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

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

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

```latex
\def\bbl@get@enc#1:#2:#3@@@{\def\bbl@hyph@enc(#2)}
```

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

\def\bbl@hook@everylanguage#1{}
\def\bbl@hook@loadpatterns#1{\input #1\relax}
\let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
\def\bbl@hook@loadkernel#1{%
\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
\expandafter\let\csname otherlanguage*\endcsname\selectlanguage
\def\bbl@usehooks##1##2{}% TODO. Temporary!!
\def\setlocale{\errhelp{Find an armchair, sit down and wait}%
\errmessage{(babel) Not yet available}}%
\let\uselocale\setlocale
\let\locale\setlocale
\let\selectlocale\setlocale
\let\localename\setlocale
\let\textlocale\setlocale
\let\textlanguage\setlocale
\let\languagetext\setlocale}
\begingroup
\def\AddBabelHook#1#2{\expandafter\ifx\csname bbl@hook@#2\endcsname\relax
\def\next{\toks1}\else
\def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}\fi
\next}
\ifx\directlua\@undefined
\ifx\XeTeXinputencoding\@undefined\else
\input xebabel.def
\fi
\else
\input luababel.def
\fi
\openin1=babel-\bbl@format.cfg
\ifeof1
\input babel-\bbl@format.cfg\relax
\else
\input babel-\bbl@format.cfg\relax
\fi
The configuration file can now be opened for reading.

The configuration file can now be opened for reading.

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

```
\def\languagename{english}\
\ifeof1
  \message{I couldn't find the file \jobname.dat, I will try the file \jobname}\relax
  \input \jobname\relax
\else
  \def\languagename{english}\relax
\fi
```

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 \texttt{-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`
```

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 reactivates the default patterns, and close the configuration file.

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

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

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

```
\let\bbl@line\undefined
\let\process@line\undefined
```
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].

\begin{verbatim}
⟨⟨∗ More package options ⟩⟩ ≡
\chardef\bbl@bidimode\z@
\DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
\DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101}
\DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102}
\DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201}
\DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202}
\DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203}
\end{verbatim}

With explicit languages, we could define the font at once, but we don’t. Just wait and see if the language is actually activated. \texttt{bb@font} replaces hardcoded font names inside \texttt{\..family} by the corresponding macro \texttt{\..default}.

At the time of this writing, fontspec shows a warning about there are languages not available, which some people think refers to babel, even if there is nothing wrong. Here is hack to patch fontspec to avoid the misleading (and mostly useless) message.

\begin{verbatim}
⟨⟨∗ Font selection ⟩⟩ ≡
\bbl@trace{Font handling with fontspec}
\ifx\ExplSyntaxOn\@undefined\else
\def\bbl@fs@warn@nx#1#2{\bbl@tempfs is the original macro
\in@{,#1,}{,no-script,language-not-exist,}\
\ifin@\else\bbl@tempfs@nx{#1}{#2}\fi}
\def\bbl@fs@warn@nxx#1#2#3{\in@{,#1,}{,no-script,language-not-exist,}\
\ifin@\else\bbl@tempfs@nxx{#1}{#2}{#3}\fi}
\def\bbl@loadfontspec{\let\bbl@loadfontspec\relax
\ifx\fontspec\@undefined
\usepackage{fontspec}\
\fi}
\fi
\@onlypreamble\babelfont
\newcommand\babelfont[2][2]{\bbl@foreach{#1}{\expandafter\ifx\csname date##1\endcsname\relax
\IfFileExists{babel-##1.tex}{\babelprovide{##1}}{}\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][2]{\bbl@ifunset{\bbl@tempb family}\
\expandafter\ifx\csname date##1\endcsname\relax
\IfFileExists{ babel-##1.tex }{ \babelprovide{##1} }{}
\fi}
\end{verbatim}

```
% For the default font, just in case:
\bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}
\expandafter\bbl@ifblank\expandafter{\bbl@tempa}{}
\let\bbl@rmdflt@\bbl@tempa\relax
\bridgefont{\bbl@rmdflt@}{\bbl@rmdflt@@}{\bbl@rmdflt@}
\foreach\bbl@font@fams{}{\bbl@exp{\bbl@rmdflt@\bbl@rmdflt@}}
\bbl@exp{\bbl@rmdflt@\bbl@rmdflt@}
\gdef\bbl@rmdflt@{\bbl@rmdflt@}
\newcommand{\textfont}{\bbl@rmdflt@}

\def\bbl@providefam#1{% Just define it
\bbl@exp{\not@math@alphabet\bbl@family\relax
\prepare@family@series@update{#1}% TODO. Fails
\fontfamily{#1default}%
\selectfont}
\DeclareTextFontCommand{\text#1}{\bbl@family}
}

\def\bbl@nostdfont#1{% Flag, to avoid dupl warns
\bbl@info{The current font is not a babel standard family:\% #1\% \fontname\font\% There is nothing intrinsically wrong with this warning, and\% you can ignore it altogether if you do not need these\% families. But if they are used in the document, you should be\% aware 'babel' will not set Script and Language for them, so\% you may consider defining a new family with \string\babelfont.\% See the manual for further details about \string\babelfont.\% Reported}}
\gdef\bbl@switchfont{\bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}
\expandafter\lowercase\expandafter{\edef\bbl@tempa{\bbl@cl{sname}}}}
\foreach\bbl@font@fams{}{\bbl@exp{\bbl@rmdflt@\bbl@rmdflt@\bbl@tempa}}
\bbl@exp{\bbl@rmdflt@\bbl@rmdflt@\bbl@tempa}}
\def\bbl@tempa{\bbl@nostdfont{}}% TODO. Don't use \bbl@tempa
\foreach\bbl@font@fams{}{\bbl@exp{\bbl@rmdflt@\bbl@rmdflt@}}
\gdef\bbl@switchfont{\bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}
\expandafter\lowercase\expandafter{\edef\bbl@tempa{\bbl@cl{sname}}}}
\foreach\bbl@font@fams{}{\bbl@exp{\bbl@rmdflt@\bbl@rmdflt@\bbl@tempa}}
\bbl@exp{\bbl@rmdflt@\bbl@rmdflt@\bbl@tempa}}
\def\bbl@tempa{\bbl@nostdfont{}}% 1=T - language, already defined
\def\bbl@exp{\bbl@tempa{\bbl@rmdflt@}}% TODO. Don't use \bbl@tempa
\foreach\bbl@font@fams{}{\bbl@exp{\bbl@rmdflt@}}
\def\bbl@tempa{\bbl@nostdfont{}}% 1=T - language, already defined
\def\bbl@exp{\bbl@tempa{\bbl@rmdflt@}}% TODO. But sometimes wrong!
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. For historical reasons, \TeX can select two different series (bx and b), for what is conceptually a single one. This can lead to problems when a single family requires several fonts, depending on the language, mainly because ‘substitutions’ with some combinations are not done consistently – sometimes bx/sc is the correct font, but sometimes points to b/n, even if b/sc exists. So, some substitutions are redefined (in a somewhat hackish way, by inspecting if the variant declaration contains >ssub*).
\def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdefault@lang \opt \fnt \nme \xxfamily
 \let\bbl@tempe\bbl@mapselect
 \edef\bbl@tempb{\bbl@stripslash#4/}% Catcodes hack (better pass it).
 \bbl@exp{\\bbl@replace\\bbl@temp\\bbl@stripslash\family/}{}}
 \let\bbl@mapselect\relax
 \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
 \let#4\@empty % Make sure \renewfontfamily is valid
 \bbl@exp{
 \let\bbl@temp@pfam<\bbl@stripslash#4 \space>% eg, '\rmfamily '
 <keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@c{sname}}{
 \newfontscript{\bbl@c{sname}}{\bbl@c{sotf}}}%
 <keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@c{lname}}{
 \newfontlanguage{\bbl@c{lname}}{\bbl@c{lotf}}}%
 \let\bbl@tempfs@nx<__fontspec_warning:nx>%
 \let<__fontspec_warning:nx>\bbl@tempfs@nx%
 \let<__fontspec_warning:nxx>\bbl@tempfs@nxx%
 \renewfontfamily\#4%
 |[\bbl@c{lsys}], % xetex removes unknown features :-(
 |ifcase\bbl@engine|or RawFeature={family=\bbl@tempb},\fi
 #2}{#3}% ie \bbl@exp{[..]{#3}
 \bbl@exp{\let<\bbl@stripslash#4 \space>}
 \bbl@temp@pfam
 \let\bbl@mapselect\bbl@tempe}

\font@rst and \font@rst@fam are only used when there is no global settings, to save and restore de previous families. Not really necessary, but done for optimization.
\def\bbl@font@rst@fams{rm,sf,tt}
\langle\langle Font selection\rangle\rangle

\textbf{9 Hooks for XeTeX and LuaTeX}

\subsection*{9.1 XeTeX}

Unfortunately, the current encoding cannot be retrieved and therefore it is reset always to utf8, which seems a sensible default.
\def\bbl@font@rst@fams{rm,sf,tt}
\langle\langle Font selection\rangle\rangle

\section{Footnote changes}
\def\bbl@trace{Bidi footnotes}
\ifnum\bbl@bidimode>\z@ % Any bidi=
 \bbl@font@rst@fams{rm,sf,tt}
\langle\langle Footnote changes\rangle\rangle
Now, the code.
\begin{Verbatim}
{\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}
\def\bbl@intrapenalty#1\@@{\
\bbl@csarg\gdef{xeipn@\languagename}{\XeTeXlinebreakpenalty #1\relax}}
\def\bbl@provide@intraspace{\
\bbl@xin@{/s}{/\bbl@cl{lnbrk}}\ifin@\else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi\ifin@
\bbl@ifunset{bbl@intsp@\languagename}{}\else{\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else\ifx\bbl@KVP@intraspace\@nnil\bbl@exp{\bbl@intraspace\bbl@cl{intsp}\@@}\fi\ifx\bbl@KVP@intrapenalty\@nnil\bbl@intrapenalty0\@@\fi\fi\ifx\bbl@KVP@intraspace\@nnil\else\expandafter\bbl@intraspace\bbl@KVP@intraspace\@@\fi\ifx\bbl@KVP@intrapenalty\@nnil\else\expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@\fi\bbl@exp{\bbl@add<extras\languagename>{\XeTeXlinebreaklocale "\bbl@cl{tbcp}" <\bbl@xeisp@\languagename> <\bbl@xeipn@\languagename>}\bbl@toglobal<extras\languagename>\bbl@add<noextras\languagename>{\XeTeXlinebreaklocale ""}\bbl@toglobal<noextras\languagename>}\ifx\bbl@ispacesize\@undefined\gdef\bbl@ispacesize{\bbl@cl{xeisp}}\ifx\AtBeginDocument\@notprerr\expandafter\@secondoftwo\fi\AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}\fi}
\ifx\DisableBabelHook\@undefined\endinput\fi %%%% TODO: why
\AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
\AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
\DisableBabelHook{babel-fontspec}
⟨⟨Font selection⟩⟩
\def\bbl@provide@extra#1{}
\end{Verbatim}

10 Support for interchar

\texttt{xetex} reserves some values for CJK (although they are not set in \texttt{xelatex}), so we make sure they are skipped. Define some user names for the global classes, too.
The machinery is activated with a hook (enabled only if actually used). Here \bbl@tempc is pre-set with \bbl@usingxeclass, defined below. The standard mechanism based on \originalTeX to save, set and restore values is used. \count@ stores the previous char to be set, except at the beginning (0) and after \bbl@upto, which is the previous char negated, as a flag to mark a range.

\addb babelhook\{babel-interchar\}{beforeextras} \% 
\nameuse{bbl@xechars@\ languagename} \%
\disable babelhook\{babel-interchar\}
\protected \def\bbl@charclass#1{\% 
\ifnum\count@<\z@ \count@ -\count@ \loop \% 
\bbl@exp{\% 
\\bbl@savevariable{\XeTeXcharclass`\Uchar\count@}}% 
\XeTeXcharclass\count@ \bbl@tempc \ifnum\count@<`#1\relax \advance\count@@one \repeat \% 
\else \bbl@savevariable{\XeTeXcharclass`\#1}% 
\XeTeXcharclass`\#1 \bbl@tempc \fi \% 
\count@`\#1\relax} \%
\newcommand\bbl@ifinterchar[1]{\% 
\let\bbl@tempa@\@gobble % Assume to ignore 
edef\bbl@tempb{\zap@space#1 \@empty}% 
\ifx\bbl@kvp@interchar\@nnil\else \bbl@replace\bbl@kvp@interchar{ }{,}% 
\bbl@foreach\bbl@tempb{\% 
\bbl@xin@{,##1,}{,\bbl@kvp@interchar,}% 
\ifin@ \let\bbl@tempa\@firstofone \fi} \% 
\fi}% 
\bbl@tempa \}% 
\newcommand\bbl@ifbabelintercharT[2]{\% 
\bbl@carg@\bbl@add{bbl@icsave@\currentoption}\{\bbl@ifinterchar{#1}{#2}}% 
\newcommand\bbl@intercharclass[3]{\% 
\enable babelhook\{babel-interchar\}% 
\bbl@carg@\newXeTeXintercharclass\{xe\textclass\#2\#1\} % 
\def\bbl@tempb\#1\% 
\ifar\#1\@empty\else \ifar\#1-\% 
\bbl@upto \\else \% 
\bbl@charclass\% 
\ifcat\noexpand\#1\relax\bbl@strip\slash\#1\else\string\#1\fi\% 
\fi \% 
\expandafter\bbl@tempb \% 
\fi}% 
\bbl@ifunset{bbl@xechars@\#1} \% 
\toks@{\% 
\bbl@savevariable{\XeTeXinterchartokenstate \XeTeXinterchartokenstate\@ne \}% 
\toks@\expandafter\expandafter\expandafter{\% 
\csname bbl@xechars@\#1\endcsname}}%
And finally, the command with the code to be inserted. If the language doesn’t define a class, then use the global one, as defined above. For the definition there is an intermediate macro, which can be ‘disabled’ with \bbl@ic@<label>@<lang>.

\protected\def\bbl@ignoreinterchar{\ifnum\language=\l@nohyphenation \expandafter\@gobble \else \expandafter\@firstofone \fi}

\newcommand\babelinterchar[5][\empty]{\let\bbl@kv@label\@empty \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}} \@namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty} \{\bbl@ignoreinterchar{#5}\} \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone \bbl@exp{\textbackslash bbl@for\textbackslash bbl@tempb\{\zap@space#4 \@empty\}}{
\XeTeXinterchartoks \@nameuse{bbl@xeclass@\bbl@tempa @\bbl@ifunset{bbl@xeclass@\bbl@tempa @#2}{}{#2}} %
\@nameuse{bbl@xeclass@\bbl@tempb @\bbl@ifunset{bbl@xeclass@\bbl@tempb @#2}{}{#2}} % = \expandafter{\csname bbl@ic@\bbl@kv@label @#2\expandafter\endcsname \csname\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty@endcsname}}}

\DeclareRobustCommand\enablelocaleinterchar[1]{\bbl@ifunset{bbl@ic@#1@\languagename}\bbl@error{unknown-interchar-b}{#1}{}{}}\bbl@csarg\let{ic@#1@\languagename}\@gobble}

\Providecommand\bbl@provide@intraspace{}
\bbl@trace{Redefinitions for bidi layout}
\def\bbl@sspre@caption{% TODO: Unused!}

\bbl@csarg\edef{xechars@#1}{% \the\toks@ \bbl@usingxeclass\csname bbl@xeclass@#2@#1\endcsname \bbl@tempb#3\@empty}}
\protected\def\bbl@usingxeclass#1{\count@\z@ \let\bbl@tempc#1}
\protected\def\bbl@upto{\ifnum\count@>\z@ \advance\count@\@ne \count@-\count@ \else\ifnum\count@=\z@ \bbl@charclass{-} \else \bbl@error{double-hyphens-class}{}{}{} \fi\fi}

\Sspre@caption is as a shorthand for tex–xet babel, which is the bidi model in both pdftex and xetex.

\providecommand{\bbl@provide@intraspace}{}
\bbl@trace{Redefinitions for bidi layout}
\def\bbl@sspre@caption{% TODO: Unused!}

10.1 Layout

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titleps, and geometry. \bbl@startskip and \bbl@endskip are available to package authors. Thanks to the \TeX expansion mechanism the following constructs are valid: \adim\bbl@startskip, \advance\bbl@startskip\adim, \bbl@startskip\adim.

Consider txtbabel as a shorthand for tex–xet babel, which is the bidi model in both pdftex and xetex.
Implicitly reverses sectioning labels in bidi=basic, because the full stop is not in contact with L numbers any more. I think there must be a better way.

\IfBabelLayout{counters*}%
\AddToHook{shipout/before}{%
  \let\bbl@tempa\bbl@opt@layout{.counters.}
  \AddToHook{shipout/after}{%
    \let\thepage\bbl@save@thepage}}{}

\IfBabelLayout{counters}%
{\let\bbl@latinarabic=\@arabic
  \def\@arabic#1{\bbl@latinarabic\@elt{#1}}%
  \let\bbl@asciiroman=\@Roman
  \def\@roman#1{\ensureascii{\bbl@asciiroman\@elt{#1}}}%
}
\fi % end if layout

\IfBabelLayout{counters}{}
⟨/xetex|texxet⟩

10.2 8-bit TeX

Which start just above, because some code is shared with xetex. Now, 8-bit specific stuff. If just one encoding has been declared, then assume no switching is necessary (1).

\def\bbl@provide@extra#1{%
\if\bbl@encoding@select@off\empty\else
\bbl@ifunset{encoding@#1}{}{%
  \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
  \count@\z@
  \bbl@foreach\bbl@tempe{%
    \def\bbl@tempd{##1}% Save last declared
    \advance\count@\@ne}%
  \ifnum\count@>\@ne % (1)
    \getlocaleproperty\bbl@tempa{identification/encodings}%
    \if\bbl@tempa\empty\fi
  \bbl@replace\bbl@tempa{ }{,}%
  \bbl@xin{,\bbl@tempd,}{,\bbl@tempe,}%
  \if\bbl@tempe\else % if main encoding included in ini, do nothing
    \bbl@exp{%
      \global\bbl@encoding@#1{\originalTeX{\selectfont}\
        \fontencoding{\bbl@tempb}\
        \selectfont}}%
      \global\bbl@encoding@#1{\originalTeX{\selectfont}\
    \fontencoding{\bbl@tempb}\
    \selectfont}}%
    \fi
  \fi}
\fi}
10.3 LuaTeX

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

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

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

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

As of 1.13, LuaTeX is taken into account. Formerly, loading of patterns on the fly didn’t work in this format, but with the new loader it does. Unfortunately, the format is not based on babel, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format \texttt{language.dat} is used (under the principle of a single source), instead of \texttt{language.def}.

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

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

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

\begin{verbatim}
110
\fi
\endgroup
\toks@{}
\count@z@ % 0=start, 1=0th, 2=normal
\def\bbl@process@line#1#2 #3 #4 {%
  \ifx=#1%
  \bbl@process@synonym{#2}%
  \else
  \bbl@process@language{#1#2}{#3}{#4}%
  \ignorespaces
\endgroup
\def\bbl@manylang{%
  \ifnum\bbl@last>\@ne
  \bbl@info{Non-standard hyphenation setup}%
  \fi
\def\bbl@manylang{%
  \let\bbl@manylang\relax
  \def\bbl@process@language{#1#2}{#3}{#4}%
  \ifcase\count@
  \@ifundefined{zth@#1}{\count@\tw@}{\count@\one}%
  \or
\end{verbatim}
\count@\tw@
\ifnum\count@=\tw@
\expandafter\addlanguage\csname l@#1\endcsname
\language\allocationnumber
\chardef\bbl@last\allocationnumber
\let\bbl@elt\relax
\xdef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
\fi
\the\toks@
\toks@{}%}
\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}{}}
global\namedef{bbl@hyphendata@0}{{hyphen.tex}{}}
\bbl@process@synonym@aux{english}{0}{hyphen.tex}{}%}
\bbl@process@synonym@aux{USenglish}{0}{}{}
\else
\global\let\bbl@languages@format\bbl@languages
\def\bbl@elt#1#2#3#4{% Remove all except language 0
\ifnum#2>\z@else
\noexpand\bbl@elt{#1}{#2}{#3}{#4}%
\fi%
\xdef\bbl@languages{\bbl@languages}%
\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
\ifx\bbl@line\empty\else
\edef\bbl@line{\bbl@line\space\space\space}%
\expandafter\bbl@process@line\bbl@line\relax
\fi
\fi
\repeat
\fi
\closein\bbl@readstream
\endgroup
\bbl@trace{Macros for reading patterns files}
\def\process@line####1####2 ####3 ####4 {}}
\AddBabelHook{luatex}{loadpatterns}{% 
  \input #1\relax 
  \expandafter\gdef\csname bbl@hyphendata@\the\language\endcsname {#1{}}}
\AddBabelHook{luatex}{loadexceptions}{% 
  \input #1\relax 
  \def\bbl@tempb##1##2{{##1}{#1}}% 
  \expandafter\xdef\csname bbl@hyphendata@\the\language\endcsname 
  {\expandafter\expandafter\expandafter\bbl@tempb 
    \csname bbl@hyphendata@\the\language\endcsname}}
\endinput\fi
% Here stops reading code for hyphen.cfg
% The following is read the 2nd time it's loaded
% First, global declarations for lua
\begingroup % TODO - to a lua file
\catcode`%=12
\catcode`\'=12
\catcode`\"=12
\catcode`\:=12
\directlua{
  Babel = Babel or {}
  function Babel.lua_error(e, a)
    tex.print([[\noexpand\csname bbl@error\endcsname{\textcolor{red}{}} .. e .. '}' .. (a or '') .. '}{{}}')
  end
  function Babel.bytes(line)
    return line:gsub("(\.)", function (chr) return unicode.utf8.char(string.byte(chr)) end)
  end
  function Babel.begin_process_input()
    if luatexbase and luatexbase.add_to_callback then
      luatexbase.add_to_callback('process_input_buffer', Babel.bytes, 'Babel.bytes')
    else
      Babel.callback = callback.find('process_input_buffer')
      callback.register('process_input_buffer', Babel.callback)
    end
  end
  function Babel.end_process_input ()
    if luatexbase and luatexbase.remove_from_callback then
      luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
    else
      callback.register('process_input_buffer',Babel.callback)
    end
  end
  function Babel.addpatterns(pp, lg)
    local lg = lang.new(lg)
    local pats = lang.patterns(lg) or ''
    lang.clear_patterns(lg)
    for p in pp:gmatch('[^%s]+') do
      ss = ''
      for i in string.utfcharacters(p:gsub('%d', '')) do
        ss = ss .. '%d?' .. i
      end
      ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
      ss = ss:gsub('%.%%d%?$', '%%.')
      pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ') 
      if n == 0 then
        tex.sprint([[\string\csname\space bbl@info\endcsname(\textcolor{red}{New pattern: \textcolor{blue}{}} .. p .. '})
      end
    end
  end
end

Babel = Babel or {}
function Babel.lua_error(e, a)
  tex.print([[\noexpand\csname bbl@error\endcsname{\textcolor{red}{}} .. e .. '}' .. (a or '') .. '}{{}}')
end
function Babel.bytes(line)
  return line:gsub("(\.)", function (chr) return unicode.utf8.char(string.byte(chr)) end)
end
function Babel.begin_process_input()
  if luatexbase and luatexbase.add_to_callback then
    luatexbase.add_to_callback('process_input_buffer', Babel.bytes, 'Babel.bytes')
  else
    Babel.callback = callback.find('process_input_buffer')
    callback.register('process_input_buffer',Babel.callback)
  end
end
function Babel.end_process_input ()
  if luatexbase and luatexbase.remove_from_callback then
    luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
  else
    callback.register('process_input_buffer',Babel.callback)
  end
end
function Babel.addpatterns(pp, lg)
  local lg = lang.new(lg)
  local pats = lang.patterns(lg) or ''
  lang.clear_patterns(lg)
  for p in pp:gmatch('[^%s]+') do
    ss = ''
    for i in string.utfcharacters(p:gsub('%d', '')) do
      ss = ss .. '%d?' .. i
    end
    ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
    ss = ss:gsub('%.%%d%?$', '%%.')
    pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ') 
    if n == 0 then
      tex.sprint([[\string\csname\space bbl@info\endcsname(\textcolor{red}{New pattern: \textcolor{blue}{}} .. p .. '})
    end
  end
end

Babel = Babel or {}
function Babel.lua_error(e, a)
  tex.print([[\noexpand\csname bbl@error\endcsname{\textcolor{red}{}} .. e .. '}' .. (a or '') .. '}{{}}')
end
function Babel.bytes(line)
  return line:gsub("(\.)", function (chr) return unicode.utf8.char(string.byte(chr)) end)
end
function Babel.begin_process_input()
  if luatexbase and luatexbase.add_to_callback then
    luatexbase.add_to_callback('process_input_buffer', Babel.bytes, 'Babel.bytes')
  else
    Babel.callback = callback.find('process_input_buffer')
    callback.register('process_input_buffer',Babel.callback)
  end
end
function Babel.end_process_input ()
  if luatexbase and luatexbase.remove_from_callback then
    luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
  else
    callback.register('process_input_buffer',Babel.callback)
  end
end
function Babel.addpatterns(pp, lg)
  local lg = lang.new(lg)
  local pats = lang.patterns(lg) or ''
  lang.clear_patterns(lg)
  for p in pp:gmatch('[^%s]+') do
    ss = ''
    for i in string.utfcharacters(p:gsub('%d', '')) do
      ss = ss .. '%d?' .. i
    end
    ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
    ss = ss:gsub('%.%%d%?$', '%%.')
    pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ') 
    if n == 0 then
      tex.sprint([[\string\csname\space bbl@info\endcsname(\textcolor{red}{New pattern: \textcolor{blue}{}} .. p .. '})
  end
113
else
    tex.sprint(\[[\text{\texttt{\csname\space bbl\@info\endcsname{Renew pattern: \}}}]} .. p .. [\text{\texttt{\}}}])
end

lang.patterns(lg, pats)
end

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

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

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

\ifx\newattribute\undefined\else % Test for plain
\newattribute\bbl\@attr\@local\e
\directlua{Babel\attr\@local\e \luatexbase\registernumber\bbl\@attr\@local\e}
\AddBabelHook\luatex{beforeextras}{%\AddBabelHook\luatex{encodedcommands}%
\fi
\edef\BabelStringsDefault{unicode}
\let\luabbl@stop\relax
\AddBabelHook\luatex{encodedcommands}{
This macro adds patterns. Two macros are used to store them: \bbl@patterns@ for the global ones and \bbl@patterns@<lang> for language ones. We make sure there is a space between words when multiple commands are used.
10.4 Southeast Asian scripts

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

\def\bbl@intraspace{&
\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{}\directlua{
Babel = Babel or {}
Babel.intrapenalties = Babel.intrapenalties or {}
Babel.intrapenalties['\csname bbl@sbcp@\languagename\endcsname'] = #1
Babel.locale_props[\the\localeid].intrapenalty = #1
}}
\begingroup
\catcode`%=12
\catcode`&=14
\catcode`\'=12
\catcode`\~=12
\gdef\bbl@seaintraspace{&
\let\bbl@seaintraspace\relax
\directlua{
Babel = Babel or {}
Babel.locale = {} % Free to use, indexed by \localeid
function Babel.linebreaking.add_before(func, pos)
tex.print(\begin{verbatim}
if pos == nil then
  table.insert(Babel.linebreaking.before, func)
else
  table.insert(Babel.linebreaking.before, pos, func)
end
}\end{verbatim}
end
function Babel.linebreaking.add_after(func)
tex.print(\begin{verbatim}
table.insert(Babel.linebreaking.after, func)
\end{verbatim}
end
\def\bbl@ltempb{\zap@space#1 \@empty}\
\def\bbl@for\bbl@tempa\bbl@tempb{%
  \bbl@fixname\bbl@tempa
  \bbl@iflanguage\bbl@tempa{%}
    \@ifundefined{bbl@patterns@\bbl@tempa}{\empty}{
      \csname bbl@patterns@\bbl@tempa\endcsname\space}{}
  \fi}}
\edef\bbl@tempb{\zap@space#1 \@empty}\
\bbl@for\bbl@tempa\bbl@tempb{%
  \bbl@fixname\bbl@tempa
  \bbl@iflanguage\bbl@tempa{%}
    \@ifundefined{bbl@patterns@\bbl@tempa}{\empty}{
      \csname bbl@patterns@\bbl@tempa\endcsname\space}{}
  \fi}}
Babel.sea_enabled = true
Babel.sea_ranges = Babel.sea_ranges or {}
function Babel.set_chranges (script, chrng)
    local c = 0
    for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
        Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
        c = c + 1
    end
end

function Babel.sea_disc_to_space (head)
local sea_ranges = Babel.sea_ranges
local last_char = nil
local quad = 655360 &% 10 pt = 655360 = 10 * 6536
for item in node.traverse(head) do
    local i = item.id
    if i == node.id'glyph' then
        last_char = item
    elseif i == 7 and item.subtype == 3 and last_char
        and last_char.char > 0x0C99 then
        quad = font.getfont(last_char.font).size
        for lg, rg in pairs(sea_ranges) do
            if last_char.char > rg[1] and last_char.char < rg[2] then
                lg = lg:sub(1, 4) &% Remove trailing number of, eg, Cyrl1
                local intraspace = Babel.intraspaces[lg]
                local intrapenalty = Babel.intrapenalties[lg]
                local n
                if intrapenalty ~= 0 then
                    n = node.new(14, 0) &% penalty
                    n.penalty = intrapenalty
                    node.insert_before(head, item, n)
                end
                n = node.new(12, 13) &% (glue, spaceskip)
                node.setglue(n, intraspace.b * quad,
                                intraspace.p * quad,
                                intraspace.m * quad)
                node.insert_before(head, item, n)
                node.remove(head, item)
            end
        end
    end
end
end
end

\bbl@luahyphenate

\begin{table}
\centering
\begin{tabular}{|c|c|c|}
\hline
\textbf{Character} & \textbf{Intraspace} & \textbf{Intrapenalty} \\
\hline
\texttt{a} & 1.5 & 2.0 \\
\texttt{b} & 2.5 & 3.0 \\
\texttt{c} & 3.5 & 4.0 \\
\hline
\end{tabular}
\caption{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.}
\end{table}

\begin{verbatim}
\catcode`%=14
\def\bbl@cjkintraspace{%\let\bbl@cjkintraspace\relax
\directlua{
  Babel = Babel or {}
  require('babel-data-cjk.lua')
  Babel.cjk_enabled = true
  function Babel.cjk_linebreak(head)
    local Glyph = node.id'glyph'
    local last_char = nil
    end
end
end
end
\end{verbatim}

\subsection{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.
local quad = 655360  % 10 pt = 655360 = 10 * 65536
local last_class = nil
local last_lang = nil

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

\bbl@luahyphenate
\gdef\bbl@luahyphenate{\let\bbl@luahyphenate\relax
\directlua{
luatexbase.add_to_callback('hyphenate',
function (head, tail)
  if Babel.linebreaking.before then
5737    for k, func in ipairs(Babel.linebreaking.before) do
5738       func(head)
5739    end
5740    end
5741    if Babel.cjk_enabled then
5742       Babel.cjk_linebreak(head)
5743    end
5744    lang.hyphenate(head)
5745    if Babel.linebreaking.after then
5746       for k, func in ipairs(Babel.linebreaking.after) do
5747          func(head)
5748       end
5749    end
5750    if Babel.sea_enabled then
5751       Babel.sea_disc_to_space(head)
5752    end
5753    end,
5754    'Babel.hyphenate')
5755  }
5756
5757\endgroup
5758\let\bbl@provide@intraspace=\iffalse\else\fi\let\bbl@ifunset=\iffalse\else\fi
5759\iffalse\else\fi\iffalse\else\fi{\iffalse\else\fi\iffalse\else\fi}
5760\iffalse\else\fi\iffalse\else\fi
5761\iffalse\else\fi\iffalse\else\fi
5762\iffalse\else\fi\iffalse\else\fi
5763\iffalse\else\fi\iffalse\else\fi
5764\iffalse\else\fi\iffalse\else\fi
5765\iffalse\else\fi\iffalse\else\fi
5766\iffalse\else\fi\iffalse\else\fi
5767\iffalse\else\fi\iffalse\else\fi
5768\iffalse\else\fi\iffalse\else\fi
5769\iffalse\else\fi\iffalse\else\fi
5770\iffalse\else\fi\iffalse\else\fi
5771\iffalse\else\fi\iffalse\else\fi
5772\iffalse\else\fi\iffalse\else\fi
5773\iffalse\else\fi\iffalse\else\fi
5774\iffalse\else\fi\iffalse\else\fi
5775\iffalse\else\fi\iffalse\else\fi
5776\iffalse\else\fi\iffalse\else\fi
5777\iffalse\else\fi\iffalse\else\fi
5778\iffalse\else\fi\iffalse\else\fi
5779\iffalse\else\fi\iffalse\else\fi
5780\iffalse\else\fi\iffalse\else\fi
5781\iffalse\else\fi\iffalse\else\fi
5782\iffalse\else\fi\iffalse\else\fi
5783\iffalse\else\fi\iffalse\else\fi
5784\iffalse\else\fi\iffalse\else\fi
5785\iffalse\else\fi\iffalse\else\fi
5786\iffalse\else\fi\iffalse\else\fi
5787\iffalse\else\fi\iffalse\else\fi
5788\iffalse\else\fi\iffalse\else\fi
5789\iffalse\else\fi\iffalse\else\fi
5790\iffalse\else\fi\iffalse\else\fi
5791\iffalse\else\fi\iffalse\else\fi
5792\iffalse\else\fi\iffalse\else\fi
5793\iffalse\else\fi\iffalse\else\fi
5794\iffalse\else\fi\iffalse\else\fi

10.6 Arabic justification

WIP. \bbl@arabicjust is executed with both elongated an kashida. This must be fine tuned. The attribute kashida is set by transforms with kashida-
5790 \iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi
5791\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi
5792 \iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi
5793 \iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi
5794 \iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi\iffalse\else\fi

119
\begin{itemize}
\item \texttt{\edef\bblar@elongated{}}
\item \texttt{\edef\bblar@nofswarn{\edef\msg_warning:nnx##1##2##3{}}}
\item \texttt{\catcode`_=11 \catcode`:=11}
\item \texttt{\gdef\bblar@kashida}{\z@}
\end{itemize}

\begin{itemize}
\item \texttt{\bblar@fetchjalt#1#2#3#4{}}
\item \texttt{\bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}}% Alef maksura
\item \texttt{\bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}}% Yeh
\end{itemize}

\begin{itemize}
\item \texttt{\addfontfeature{RawFeature=+jalt}}% @namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
\end{itemize}

Elongated forms. Brute force. No rules at all, yet. The ideal: look at jalt table. And perhaps other tables (falt?, cswh?). What about kaf? And diacritic positioning?

\begin{itemize}
\item \texttt{\edef\bblar@parsejalti{}}
\end{itemize}

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

\begin{itemize}
\item \texttt{\edef\bblar@parsejalti{}}
\end{itemize}

\begin{itemize}
\item \texttt{\edef\bblar@parsejalti{}}
\end{itemize}

\texttt{\def\bblar@fetchjalt#1#2#3#4{}}
\texttt{\bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}}% Alef maksura
\texttt{\bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}}% Yeh

\texttt{\addfontfeature{RawFeature=+jalt}}% @namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
\directlua{
for k, v in pairs(Babel.arabic.from) do
    if Babel.arabic.dest[k] and
        not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
        Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
        [Babel.arabic.from[k]] = Babel.arabic.dest[k]
    end
end
}

The actual justification (inspired by \textit{chickenize}).

\begingroup
\catcode`#=11
\catcode`~=11
\directlua{
Babel.arabic = Babel.arabic or {}
Babel.arabic.from = {}
Babel.arabic.dest = {}
Babel.arabic.justify_factor = 0.95
Babel.arabic.justify_enabled = true
Babel.arabic.kashida_limit = -1

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

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

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

    if line == nil then
        line = {}
        line.glue_sign = 1
    end
}
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.
    % Elongated glyphs
    if elong_map then
      local locale = node.get_attribute(n, LOCALE)
      if elong_map[locale] and elong_map[locale][n.font] and
        elong_map[locale][n.font][n.char] then
        table.insert(elongs, {node = n, locale = locale} )
        node.set_attribute(n.prev, KASHIDA, 0)
      end
    end
  end % of node.traverse_id
end % Tatwil
if Babel.kashida_wts then
  local k_wt = node.get_attribute(n, KASHIDA)
  if k_wt > 0 then % todo. parameter for multi inserts
    table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
  end
end
end % of node.traverse_id

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

% == Elongated ==
% Original idea taken from 'chikenize'
while (#elongs > 0 and width < goal) do
  subst_done = true
  local x = #elongs
  local curr = elongs[x].node
  local oldchar = curr.char
  curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
  width = node.dimensions(line.head) % Check if the line is too wide
  if width > goal then
    curr.char = oldchar
    break
  end
  if #elongs == 0 then goto next_line end
end % Tatwil ==
width = node.dimensions(line.head)  \% The 'natural' width
k_curr = \#k_list \% Traverse backwards, from the end
wt_pos = 1

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

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

% Must take into account marks and ins, see luatex manual.
% Have to be executed only if there are changes. Investigate
% what's going on exactly.
if subst_done and not gc then
  d = node.hpack(line.head, full, 'exactly')
  d.shift = shift
  node.insert_before(head, line, d)
  node.remove(head, line)
end
end % if process line
endgroup
\ifnum\bbl@bidimode>100...
10.7 Common stuff

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

⟨⟨ Font selection ⟩⟩

10.8 Automatic fonts and ids switching

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

% TODO - to a lua file
\directlua{
Babel.script\_blocks = {
  ['dflt'] = {},
  ['Arab'] = {{0x0600, 0x06FF}, {0x0800, 0x08FF}, {0x0750, 0x077F},
              {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
  ['Armn'] = {{0x0530, 0x058F}},
  ['Beng'] = {{0x0980, 0x09FF}},
  ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
  ['Cyril'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
               {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
  ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
  ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
              {0x102E0, 0x102FF}},
  ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
  ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
              {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FF},
              {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
              {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
              {0x2C60, 0x2C7F}, {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
  ['Hebr'] = {{0x0590, 0x05FF}},
  ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
              {0x4E00, 0x9FF}, {0xFF00, 0xFFEF}},
  ['Knda'] = {{0x0C80, 0x0CFF}},
  ['Kore'] = {{0x1100, 0x11FF}, {0x19E0, 0x19FF}},
  ['Konk'] = {{0x08F0, 0x08FF}},
  ['Kr' }
function Babel.locale_map(head)
  if not Babel.locale_mapped then return head end
  local LOCALE = Babel.attr_locale
  local GLYPH = node.id('glyph')
  local inmath = false
  local toloc_save
  for item in node.traverse(head) do
    local toloc
    if not inmath and item.id == GLYPH then
      % Optimization: build a table with the chars found
      if Babel.chr_to_loc[item.char] then
        toloc = Babel.chr_to_loc[item.char]
      else
        for lc, maps in pairs(Babel.loc_to_scr) do
          for _, rg in pairs(maps) do
            if item.char >= rg[1] and item.char <= rg[2] then
              Babel.chr_to_loc[item.char] = lc
              toloc = lc
              break
            end
          end
        end
      end
    end
  end
  if toloc == -2000 then
    toloc = toloc_save
  elseif toloc == -1000 then
    toloc = nil
  end
  if toloc and Babel.locale_props[toloc].lg then
    item.lang = Babel.locale_props[toloc].lg
  end
end
The code for \bblcharproperty is straightforward. Just note the modified lua table can be different.

```latex
\newcommand\bblcharproperty[1]{%
  \count@=#1\relax
  \ifvmode
    \expandafter\bbl@chprop
  \else
    \bbl@error{charproperty-only-vertical}{\ifnum\count@=0}{}{}% 
  \fi}
\newcommand\bbl@chprop[3][\the\count@]{% 
  \@tempcnta=#1\relax
  \bbl@ifunset{bbl@chprop@#2}% {unknown-char-property}
  {\bbl@error{unknown-char-property}{\ifnum\count@=0}{#2}{}}%
  {%
    \loop
    \bbl@cs{chprop@#2}{#3}%
    \ifnum\count@<\@tempcnta
      \advance\count@\@ne
    \repeat}
  }% 
  \def\bbl@chprop@direction#1{%
    \directlua{
      Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
      Babel.characters[\the\count@]['d'] = '#1'
    }
  }% 
  \let\bbl@chprop@bc\bbl@chprop@direction
  \def\bbl@chprop@mirror#1{%
    \directlua{
      Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
      Babel.characters[\the\count@]['m'] = '\number#1'
    }
  }% 
  \let\bbl@chprop@bmg\bbl@chprop@mirror
  \def\bbl@chprop@linebreak#1{%
    \directlua{
      Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
      Babel.cjk_characters[\the\count@]['c'] = '#1'
    }
  }% 
  \let\bbl@chprop@lb\bbl@chprop@linebreak
  \def\bbl@chprop@locale#1{%
    \directlua{
      Babel.chr_to_loc = Babel.chr_to_loc or {}
      Babel.chr_to_loc[\the\count@] = \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}
    }
  }% 
  \def\bbl@chprop@direction#1{%
    \directlua{
      Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
      Babel.characters[\the\count@]['d'] = '#1'
    }
  }% 
  \let\bbl@chprop@bc\bbl@chprop@direction
  \def\bbl@chprop@mirror#1{%
    \directlua{
      Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
      Babel.characters[\the\count@]['m'] = '\number#1'
    }
  }% 
  \let\bbl@chprop@bmg\bbl@chprop@mirror
  \def\bbl@chprop@linebreak#1{%
    \directlua{
      Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
      Babel.cjk_characters[\the\count@]['c'] = '#1'
    }
  }% 
  \let\bbl@chprop@lb\bbl@chprop@linebreak
  \def\bbl@chprop@locale#1{%
    \directlua{
      Babel.chr_to_loc = Babel.chr_to_loc or {}
      Babel.chr_to_loc[\the\count@] = \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}
    }
  }% 
}\endgroup
```

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.

```lua
Babel.nohyphenation = \the\l@nohyphenation

Now the \TeX{} high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the \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 in 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 @, we just avoid this character in macro names (which explains the internal group, too).

```
\bbl@forkv{[#1]}{%
\inp{[##1]},{nil,step,remove,insert,string,no,pre,\%}
\nin{no,post,penalty,kashida,space,spacefactor,kern,node,after,\%}
\ifclose\else
  \bbl@error{bad-transform-option}{##1}{\%}{\%}
\fi\%}
\let\bbl@kv@attribute\relax
\let\bbl@kv@label\relax
\let\bbl@kv@fonts\@empty
\bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}\%
\ifx\bbl@kv@fonts\@empty\else\bbl@settransfont\fi
\ifx\bbl@kv@attribute\relax\ifx\bbl@kv@label\relax\else
  \edef\bbl@kv@fonts{\bbl@kv@fonts}\%
  \edef\bbl@kv@attribute{bbl@ATR\bbl@kv@label @#3\bbl@kv@fonts} \%
  \count@\z@
  \bbl@elt##1##2##3{\%\bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}\%\bbl@ifsamestring{\bbl@kv@fonts}{##3}\%\count@\@ne\%
  \bbl@error{font-conflict-transforms}\{}\{}}\%
  }\%
  \bbl@transfont@list\%
  \ifnum\count@=\z@
    \bbl@exp{\global\bbl@add\bbl@transfont@list{\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}\%
  \fi\%
  \bbl@ifunset{\bbl@kv@attribute}\%
  \global\bbl@carg\newattribute{\bbl@kv@attribute}\%
\else\edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}\%
\fi\%
\directlua{
  local lbkr = Babel.linebreaking.replacements[#1]
  local u = unicode.utf8
  local id, attr, label
  if #1 == 0 then
    id = \the\csname bbl@id@@#3\endcsname\space
  else
    id = \the\csname l@#3\endcsname\space
  end
  \if\bbl@kv@attribute\relax
    attr = -1
  \else
    attr = luatexbase.registernumber'\bbl@kv@attribute'\%
  \fi\%
  \if\bbl@kv@label\relax\else \% Same refs:
    label = [\=\[\bbl@kv@label]\=]\%
  \fi\%
  \directlua{
    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} })

let\bbl@transfont@list\@empty
let\bbl@transfont\relax % Execute only once
let\bbl@transfam=-unknown-
\bbl@foreach\bbl@font@fams{}
\AddToHook{##1family}{\def\bbl@transfam{##1}}
\bbl@ifsamestring{\@nameuse{##1default}}familydefault
{
\AddToHook{##1family}{\def\bbl@transfam{##1}}
}
\DeclareRobustCommand\enablelocaletransform[1]{
    \bbl@ifunset{bbl@ATR@#1@\languagename @}{}
    \bbl@csarg\setattribute{ATR@#1@\languagename @}#1}
\DeclareRobustCommand\disablelocaletransform[1]{
    \bbl@ifunset{bbl@ATR@#1@\languagename @}{}
    \bbl@csarg\unsetattribute{ATR@#1@\languagename @}#1}
\def\bbl@activateposthyphen{\let\bbl@activateposthyphen\relax}
\directlua{
    require('babel-transforms.lua')
    Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
}
\def\bbl@activateprehyphen{\let\bbl@activateprehyphen\relax}
\directlua{
    require('babel-transforms.lua')
}
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 \texttt{[==]}). 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 \texttt{lua} code is in the \texttt{lua} file below.

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

### 10.9 Bidi

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

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

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

\begin{verbatim}
\breakafterdirmode=1
\ifnum\bb@bidimode=\@ne % Any bidi= except default=1
  \let\bb@beforeforeign\leavevmode
  \AtEndOfPackage\EnableBabelHook{babel-bidi}
  \RequirePackage\luaotfload
  \bb@activate@preotf
\end{verbatim}
\directlua{require('babel-data-bidi.lua')}
\ifcase\textwidth\expandafter\@gobbletwo\the\bbl@bidimode
\or
\require('babel-bidi-basic.lua')
\or
\require('babel-bidi-basic-r.lua')\fi}
\newattribute\bbl@attr@dir
\directlua{ Babel.attr_dir = luatexbase.registernumber'\bbl@attr@dir' }
\bbl@exp{\output{\bodydir\pagedir\the\output}}
\chardef\bbl@thetextdir\z@
\chardef\bbl@thepardir\z@
\def\bbl@getluadir#1{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
\ifcase#3\relax
\ifcase\bbl@getluadir{#1}\relax
\else
\ifcase\bbl@getluadir{#1}
\else
\fi
\frozen@everymath\expandafter{\expandafter\bbl@everymath\the\frozen@everymath}
\frozen@everydisplay\expandafter{\expandafter\bbl@everydisplay\the\frozen@everydisplay}
\AtBeginDocument{
\directlua{
function Babel.math_box_dir(head)
if not (token.get_macro('bbl@insidemath') == '0') then
if Babel.hlist_has_bidi(head) then
local d = node.new(node.id'dir')
d.dir = '+TRT'
node.insert_before(head, node.has_glyph(head), d)
end
end
return d
end
\\bbl@setluadir{#1}{#2}{#3}{1=text/par.. 2=\textdir.. 3=0 lr/1 rl
\ifcase#3\relax
\ifcase\bbl@getluadir{#1}\relax
\else
\ifcase\bbl@getluadir{#1}
\else
\fi
\fi
\AtBeginDocument{
\directlua{
function Babel.math_box_dir(head)
if not (token.get_macro('bbl@insidemath') == '0') then
if Babel.hlist_has_bidi(head) then
local d = node.new(node.id'dir')
d.dir = '+TRT'
node.insert_before(head, node.has_glyph(head), d)
local inmath = false
\RTLtextinsidecmathneedspecialattention. Itaffectsnotonlytoactualmathstuff, but also to 'tabular', which is based on a fake math.
10.10 Layout

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

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

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

With the issue \#15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, \texttt{tabular} seems to work (at least in simple cases) with \texttt{array}, \texttt{tabularx}, \texttt{hhline}, \texttt{colortbl}, \texttt{longtable}, \texttt{booktabs}, etc. However, \texttt{dcolumn} still fails.
\def\bbl@provide@extra#1{% 
% == Counters: mapdigits == 
% Native digits 
\ifx\bbl@KVP@mapdigits@null\else \bbl@ifunset{bbl@dgnat@languagename}{}% \%\% -> presets in luababel 
\bbl@activate@preotf \directlua{ 
Babel = Babel or {} %%% -> presets in luababel 
Babel.digits_mapped = true 
Babel.digits = Babel.digits or {} 
Babel.digits[\the\localeid] = table.pack(string.utfvalue(\bbl@cl{dgnat}')) 
if not Babel.numbers then 
function Babel.numbers(head) 
local LOCALE = Babel.attr_locale 
local GLYPH = node.id'glyph' 
local inmath = false 
for item in node.traverse(head) do 
if not inmath and item.id == GLYPH then 
local temp = node.get_attribute(item, LOCALE) 
if Babel.digits[temp] then 
local chr = item.char 
if chr > 47 and chr < 58 then 
item.char = Babel.digits[temp][chr-47] 
end 
end 
elseif item.id == node.id'math' then 
inmath = (item.subtype == 0) 
end 
end 
end 
return head 
end 
end 
} \fi 
% == transforms == 
\ifx\bbl@KVP@transforms@null\else \def\bbl@elt##1##2##3{\in@{$transforms.}{$##1}% \ifin@ 
\def\bbl@tempa{##1}{\bbl@replace\bbl@tempa{transforms.}{}{\bbl@carg\bbl@transforms{babel\bbl@tempa}{##2}{##3}}} \fi} \csname bbl@inidata@languagename\endcsname \bbl@release@transforms \relax % \relax closes the last item. 
\fi 
% Start tabular here: 
\ifcase\bbl@thetextdir % \localerestoredirs{% 
\ifnum\textdirection=z@\else\textdir TLT\fi 
\else \ifnum\textdirection=1\else\textdir TRT\fi 
\fi 
\ifcase\bbl@thepardir % \localerestoredirs{% 
\ifnum\pardirection=z@\else\pardir TLT\bodydir TLT\fi 
\else \ifnum\pardirection=1\else\pardir TRT\bodydir TRT\fi 
\fi 
\fi 
\IfBabelLayout{tabular}% 
\chardef\bbl@tabular@mode=2% All RTL 
\IfBabelLayout{notabular}
Very likely the output routine must be patched in a quite general way to make sure the `\bodydir` is set to `\pagedir`. Note outside `\output` they can be different (and often are). For the moment, two ad hoc changes.

\AtBeginDocument{%}
\@ifpackageloaded{multicol}%
\toks0\expandafter{\multi@column@out移植\the\toks0}%
\def\multi@column@out\bodydir\pagedir\unexpanded\expandafter{\pcol@output}{%
\bodydir\pagedir\unexpanded\expandafter{\pcol@output}}%
\ifx\bbl@opt@layout\@nnil\endinput% if no layout

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{}% non-local changes, use always inside a group!
\bbl@exp{%}
\def\bbl@insidemath{0}%
"}
Babel.picture_has_bidi = 0

function Babel.picture_dir (head)
    if not Babel.get_picture_dir then return head end
    if Babel.hlist_has_bidi(head) then
        Babel.picture_has_bidi = 1
    end
    return head
end

luatexbase.add_to_callback("hpack_filter", Babel.picture_dir, "Babel.picture_dir")

\AtBeginDocument{%
\def\LS@rot{%
\setbox\@outputbox\vbox{%
    \hbox dir TLT\{\rotatebox{90}{\box\@outputbox}}}}%
\long\def\put(#1,#2)#3{%
\@killglue
% Try:
@ifx\bbl@pictresetdir\relax
  \def\bbl@tempc{0}%
  \else
  \directlua{
    Babel.get_picture_dir = true
    Babel.picture_has_bidi = 0
  }%
  \setbox\z@\hb@xt@\z@{%
    \@defaultunitsset\@tempdimc{#1}\unitlength
    \kern\@tempdimc
    #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\hb@xt\z@{%
  \@defaultunitsset\@tempdimc{#1}\unitlength
  \kern\@tempdimc
  {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
\ignorespaces}
\MakeRobust\put}%
\AtBeginDocument
{\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir\@gobble}%
  \ifx\pgfpicture\@undefined\else % TODO. Allow deactivate?
    \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
    \bbl@add\pgfinterruptpicture{\bbl@pictsetdir\z@}%
  \fi%
\ifx\tikzpicture\@undefined\else
  \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\tw@}%
  \bbl@add\tikz@atbegin@node{\bbl@pictsetdir}%
  \bbl@sreplace\tikz{\begingroup\bbl@pictsetdir\tw@}%
\fi%
\ifx\tcolorbox\@undefined\else
  \def\tcb@drawing@env@begin{%\csname tcb@before@	cb@split@state\endcsname
    \bbl@pictsetdir\tw@\begin{\kvtcb@graphenv}\tcb@bbdraw\tcb@apply@graph@patches}
  \def\tcb@drawing@env@end{%\end{\kvtcb@graphenv}\bbl@pictresetdir\csname tcb@after@	cb@split@state\endcsname}%

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

\IfBabelLayout{extras}{}
\IfBabelLayout{footnotes}{}
\IfBabelLayout{counters}{%}
\directlua{
  luatexbase.add_to_callback("process_output_buffer",
        \Babel.discard_sublr , "Babel.discard_sublr") }{}
\IfBabelLayout{counters}{%}
\IfBabelLayout{footnotes}{%}
\IfBabelLayout{extras}{}{}}{}{}}{}

10.11 Lua: transforms
After declaring the table containing the patterns with their replacements, we define some auxiliary functions: str_to_nodes converts the string returned by a function to a node list, taking the node at base as a model (font, language, etc.); fetch_word fetches a series of glyphs and discretionary, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck).
post_hyphenate_replace is the callback applied after lang.hyphenate. This means the automatic hyphenation points are known. As empty captures return a byte position (as explained in the 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.

```
6954 ⟨transforms⟩
6955 Babel.linebreaking.replacements = {}  
6956 Babel.linebreaking.replacements[0] = {} -- pre  
6957 Babel.linebreaking.replacements[1] = {} -- post
6958  
6959 -- Discretionaries contain strings as nodes
6960 function Babel.str_to_nodes(fn, matches, base)
6961     local n, head, last
6962     if fn == nil then return nil end
6963     for s in string.utfvalues(fn(matches)) do
6964         if base.id == 7 then
6965             base = base.replace
6966         end
6967         n = node.copy(base)
6968         n.char = s
6969         if not head then
6970             head = n
6971         else
6972             last.next = n
6973         end
6974         last = n
6975     end
6976     return head
6977 end
6978  
6979 Babel.fetch_subtext = {}
6980  
6981 Babel.ignore_pre_char = function(node)
6982     return (node.lang == Babel.nohyphenation)
6983 end
6984  
6985 -- Merging both functions doesn't seem feasible, because there are too many differences.
6986 Babel.fetch_subtext[0] = function(head)
6987     local word_string = ''
6988     local word_nodes = {}
6989     local lang
6990     local item = head
6991     local inmath = false
6992     while item do
6993         if item.id == 11 then
6994             inmath = (item.subtype == 0)
6995         end
6996         if inmath then
6997             -- pass
6998         elseif item.id == 29 then
6999             local locale = node.get_attribute(item, Babel.attr_locale)
7000             if lang == locale or lang == nil then
7001                 lang = lang or locale
7002                 if Babel.ignore_pre_char(item) then
7003                     word_string = word_string .. Babel.us_char
7004                 else
7005                     word_string = word_string .. unicode.utf8.char(item.char)
7006                 end
7007             end
7008         end
7009     end
7010     return word_string
7011 end
7012
```

140
```lua
    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
    elseif item.id == 7 and item.subtype == 2 then
        word_string = word_string .. '='
        word_nodes[#word_nodes+1] = item
    elseif item.id == 7 and item.subtype == 3 then
        word_string = word_string .. '|'  
        word_nodes[#word_nodes+1] = item
    -- (1) Go to next word if nothing was found, and (2) implicitly
```

-- remove leading USs.
elseif word_string == '' then
  -- pass

-- This is the responsible for splitting by words.
elseif (item.id == 12 and item.subtype == 13) then
  break
else
  word_string = word_string .. Babel.us_char
  word_nodes[#word_nodes+1] = item -- Will be ignored
end

item = item.next

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

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
      if mw == nil and w == '' then break end
      if not lang then goto next end
      if not lbkr[lang] then goto next end
      -- after the match, either as found by u.match (faster) or the
      -- This variable is set in some cases below to the first *byte*
-- 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 dummy_node -- used by after

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

  -- Get and remove empty captures (with ()'s, which return a
  -- number with the position), and keep actual captures
  -- (from (...)), if any, in matches.
  local first = table.remove(matches, 1)
  local last = table.remove(matches, #matches)
  -- Non re-fetched substrings may contain \31, which separates
  -- subsubstrings.
  if string.find(w:sub(first, last-1), Babel.us_char) then break end

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

  -- 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

  ------ TODO. dummy_node?
  while rc < last-first+1 or dummy_node do -- for each replacement
    if Babel.debug then
      print('.....', rc + 1)
    end
    sc = sc + 1
    rc = rc + 1
end
if Babel.debug then
    Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
    local ss = ''
    for itt in node.traverse(head) do
        if itt.id == 29 then
            ss = ss .. unicode.utf8.char(itt.char)
        else
            ss = ss .. '{' .. itt.id .. '}'
        end
    end
    print('***************', ss)
end

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

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

if crep then
    step = crep.step or step
end

if crep and crep.after then
    crep.insert = true
    if dummy_node then
        item = dummy_node
    else -- TODO. if there is a node after?
        d = node.copy(item_base)
        head, item = node.insert_after(head, item, d)
        dummy_node = item
    end
end

if crep and not crep.after and dummy_node then
    node.remove(head, dummy_node)
    dummy_node = nil
end

if (not enabled) or (crep and next(crep) == nil) then -- = {}
    if step == 0 then
        last_match = save_last -- Optimization
    else
        last_match = utf8.offset(w, sc+step)
    end
    goto next
end

elseif crep == nil or crep.remove then
    node.remove(head, item)
    table.remove(w_nodes, sc)
    w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
    sc = sc - 1 -- Nothing has been inserted.
    last_match = utf8.offset(w, sc+1+step)
    goto next
elseif crep and crep.kashida then -- Experimental
    node.set_attribute(item, Babel.attr_kashida, 144)
elseif crep and crep.string then
    local str = crep.string(matches)
    if str == '' then    -- Gather with nil
        node.remove(head, item)
        table.remove(w_nodes, sc)
        w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
        sc = sc - 1    -- Nothing has been inserted.
    else
        local loop_first = true
        for s in string.utfvalues(str) do
            d = node.copy(item_base)
            d.char = s
            if loop_first then
                loop_first = false
                head, new = node.insert_before(head, item, d)
            elseif 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)
            if Babel.debug then
                print('.....', 'str')
                Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
            end
            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
        node.remove(head, item)
    end -- if ''
    last_match = utf8.offset(w, sc+1+step)
    goto next
end

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)
end -- if '1'

elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
    -- ERROR

elseif crep and crep.penalty then
    d = node.new(14, 0)   -- (penalty, userpenalty)
    d.attr = item_base.attr
    d.penalty = crep.penalty
    head, new = node.insert_before(head, item, d)
end
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.norule then
    -- 655360 = 10 pt = 10 * 65536 sp
    d = node.new(2, 3) -- (rule, empty) = \no*rule
    local quad = font.getfont(item_base.font).size or 655360
    d.width = crep.norule[1] * quad
    d.height = crep.norule[2] * quad
    d.depth = crep.norule[3] * quad
    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
elseif crep and crep.kern then
    d = node.new(13, 1) -- (kern, user)
    local quad = font.getfont(item_base.font).size or 655360
    d.attr = item_base.attr
    d.kern = crep.kern * quad
    head, new = node.insert_before(head, item, d)
elseif crep and crep.node then
    d = node.new(crep.node[1], crep.node[2])
    d.attr = item_base.attr
    head, new = node.insert_before(head, item, d)
end -- ie replacement cases

-- Shared by disc, space(factor), kern, node 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

146
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
if dummy_node then
    node.remove(head, dummy_node)
    dummy_node = nil
end
end -- for match
end -- for patterns
::next::
word_head = nw
end -- for substring
return head
-- This table stores capture maps, numbered consecutively
Babel.capture_maps = {}
-- The following functions belong to the next macro
function Babel.capture_func(key, cap)
    local ret = "[" .. cap:gsub('{{[0-9]}}', ""]..m[1].."]"
    local cnt
    local u = unicode.utf8
    ret, cnt = ret:gsub('{(%x%x%x%x+)}', function (n)
        return u.char(tonumber(n, 16))
    end)
end
function Babel.capt_map(from, mapno)
    return Babel.capture_maps[mapno][from] or from
end
-- Handle the {n|abc|ABC} syntax in captures
function Babel.capture_func_map(capno, from, to)
    local u = unicode.utf8
    from = u.gsub(from, '%[\%\%\%x%]+')
    function (n)
        return u.char(tonumber(n, 16))
    end)
end
for m = 1, #capno, 1 do
    to = u.gsub(to, '%[\%\%\%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
local mlen = table.getn(Babel.capture_maps)
for s in string.utfcharacters(to) do
    Babel.capture_maps[mlen][froms[cnt]] = s
    cnt = cnt + 1
end
return "]]..Babel.capt_map(m[" .. capno .. "],"..
              (mlen) .. ").. "[["
end

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

function Babel.capture_node(id, subtype)
    local sbt = 0
    for k, v in pairs(node.subtypes(id)) do
        if v == subtype then sbt = k end
    end
    return 'node = {' .. node.id(id) .. ', ' .. sbt .. '}'
end

-- Experimental: applies prehyphenation transforms to a string (letters
-- and spaces).
function Babel.string_prehyphenation(str, locale)
    local n, head, last, res
    head = node.new(8, 0) -- dummy (hack just to start)
    last = head
    for s in string.utfvalues(str) do
        if s == 20 then
            n = node.new(12, 0)
        else
            n = node.new(29, 0)
            n.char = s
        end
        node.set_attribute(n, Babel.attr_locale, locale)
        last.next = n
        last = n
    end
    head = Babel.hyphenate_replace(head, 0)
    res = ''
    for n in node.traverse(head) do
        if n.id == 12 then
The file babel-data-bidi.lua currently only contains data. It is a large and boring file and it is not shown here (see the generated file), but here is a sample:

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

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

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

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

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

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

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

```
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
```

10.12 Lua: Auto bidi with basic and basic-r
node.insert_before(head, from, d)

d = node.new(DIR)
d.dir = '-' .. dir
node.insert_after(head, to, d)

function Babel.bidi(head, ispar)
    local first_n, last_n = first and last char with nums
    local last_es = an auxiliary 'last' used with nums
    local first_d, last_d = first and last char in L/R block
    local dir, dir_real

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

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

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

    local last_lr

    local type_n = ''

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

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

    if new_dir then
        attr_dir = 0
        for at in node.traverse(item.attr) do
            if at.number == Babel.attr_dir then
                attr_dir = at.value & 0x3
    end

    if inmath then dir = ('TRT' == tex.mathdir) and 'r' or 'l' end

if attr_dir == 1 then
    strong = 'r'
elseif attr_dir == 2 then
    strong = 'al'
else
    strong = 'l'
end
strong_lr = (strong == 'l') and 'l' or 'r'
outer = strong_lr
new_dir = false
end

if dir == 'nsm' then dir = strong end -- W1
Numbers. The dual <al>/<r> system for R is somewhat cumbersome.
dir_real = dir -- We need dir_real to set strong below
if dir == 'al' then dir = 'r' end -- W3
By W2, there are no <en> <et> <es> if strong == <al>, only <an>. Therefore, there are not <et en> nor <en et>, W5 can be ignored, and W6 applied:
if strong == 'al' then
    if dir == 'en' then dir = 'an' end -- W2
    if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
    strong_lr = 'r' -- W3
end

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

Numbers in R mode. A sequence of <en>,<et>,<an>,<es> and <cs> is typeset (with some rules) in L mode. We store the starting and ending points, and only when anything different is found (including nil, ie, a non-char), the textdir is set. This means you cannot insert, say, whatstil, 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
    type_n = dir
    end
    first_n = first_n or item
    last_n = last_es or item
    last_es = nil
elseif dir == 'es' and last_n then -- W3+W6
    last_es = item
elseif dir == 'cs' then -- it's right - do nothing
elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
    if strong_lr == 'r' and type_n ~= '' then
        dir_mark(head, first_n, last_n, 'r')
    elseif strong_lr == 'l' and first_d and type_n == 'an' then
        dir_mark(head, first_n, last_n, 'r')
        dir_mark(head, first_d, last_d, outer)
        first_d, last_d = nil, nil
    elseif strong_lr == 'l' and type_n ~= '' then
        last_d = last_n
        end
        type_n = ''
first_n, last_n = nil, nil
end

R text in L, or L text in R. Order of dir_mark's are relevant: d goes outside n, and therefore it's emitted after. See dir_mark to understand why (but is the nesting actually necessary or is a flat dir structure enough?). Only L, R (and AL) chars are taken into account – everything else, including spaces, whatstis, etc., are ignored:

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

Mirroring. Each chunk of text in a certain language is considered a “closed” sequence. If <r on r> and <l on l>, it’s clearly <r> and <l>, respety, 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
      ch.char = characters[ch.char].m or ch.char
    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

And here the Lua code for bidi=basic:

Babel = Babel or {}

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

-- To cancel mirroring. Also OML, OMS, U?
Babel.symbol_fonts = Babel.symbol_fonts or {}
Babel.symbol_fonts.font.map('tenln') = true
Babel.symbol_fonts.font.map('tenlnw') = true
Babel.symbol_fonts.font.map('tencirc') = true
Babel.symbol_fonts.font.map('tencircw') = true

Babel.bidi_enabled = true
Babel.mirroring_enabled = true
require('babel-data-bidi.lua')

local characters = Babel.characters
local ranges = Babel.ranges
local DIR = node.id('dir')
local GLYPH = node.id('glyph')

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

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

153
new_state.san, new_state.ean = nil, nil
return head, new_state
end

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

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

function Babel.bidi(head, ispar, hdir)
    local d -- d is used mainly for computations in a loop
    local prev_d = ''
    local new_d = false
    local nodes = {}
    local outer_first = nil
    local inmath = false
    local glue_d = nil
    local glue_i = nil
    local has_en = false
    local first_et = nil
    local has_hyperlink = false
    local ATDIR = Babel.attr_dir
    local save_outer
    local temp = node.get_attribute(head, ATDIR)
    if temp then
        temp = temp & 0x3
        save_outer = (temp == 0 and 'l') or
                    (temp == 1 and 'r') or
                    (temp == 2 and 'al')
    elseif ispar then -- Or error? Shouldn't happen
        save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
    else -- Or error? Shouldn't happen
        save_outer = ('TRT' == hdir) and 'r' or 'l'
    end
    -- 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
    -- 'al' is only taken into account in the first, current loop
    if save_outer == 'al' then save_outer = '' end
    local fontmap = Babel.fontmap
    for item in node.traverse(head) do
-- In what follows, #node is the last (previous) node, because the
-- current one is not added until we start processing the neutrals.

-- three cases: glyph, dir, otherwise
if glyph not symbol_font(item)
or (item.id == 7 and item.subtype == 2) then
  local d_font = nil
  local item_r
  if item.id == 7 and item.subtype == 2 then
    item_r = item.replace -- automatic discs have just 1 glyph
  else
    item_r = item
  end

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

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

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

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

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

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

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

-- ET/EN -- W5 + W7->l / W6->on
if d == 'et' then
    first_et = first_et or (#nodes + 1)
elseif d == 'en' then
    has_en = true
    first_et = first_et or (#nodes + 1)
elseif first_et then -- d may be nil here!
    if has_en then
        if last == 'l' then
            temp = 'l' -- W7
else
    temp = 'en' -- W5
end
else
    temp = 'on' -- W6
end
for e = first_et, #nodes do
    if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
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
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
end
for e = first_et, #nodes do
    if glyph_not_symbol_font(nodes[e][1]) 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 --------------
outert = save_outer
last = outer
local first_on = nil
for q = 1, #nodes do
    local item
local outer_first = nodes[q][3]
outer = outer_first or outer
last = outer_first or last

local d = nodes[q][2]
if d == 'an' or d == 'en' then d = 'r' end
if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end

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

-------------- IMPLICIT, REORDER ----------------
outer = save_outer
last = outer

local state = {}
state.has_r = false

for q = 1, #nodes do
  local item = nodes[q][1]
  outer = nodes[q][3] or outer
  local d = nodes[q][2]
  if d == 'nsm' then d = last end -- W1
  if d == 'en' then d = 'an' end
  local isdir = (d == 'r' or d == 'l')
  if outer == 'l' and d == 'an' then
    state.san = state.san or item
    state.ean = item
  elseif state.san then
    head, state = insert_numeric(head, state)
  end
  for r = first_on, q - 1 do
    nodes[r][2] = temp
    item = nodes[r][1]
    if Babel.mirroring_enabled and glyph_not_symbol_font(item)
      and temp == 'r' and characters[item.char] then
      local font_mode = ''
      if item.font > 0 and font.fonts[item.font].properties then
        font_mode = font.fonts[item.font].properties.mode
      end
      if font_mode ~= 'harf' and font_mode ~= 'plug' then
        item.char = characters[item.char].m or item.char
      end
    end
  end
  first_on = nil
end

-------------- IMPLICIT, REORDER ----------------
outer = save_outer
last = outer

local state = {}
state.has_r = false

for q = 1, #nodes do
  local item = nodes[q][1]
  outer = nodes[q][3] or outer
  local d = nodes[q][2]
  if d == 'nsm' then d = last end -- W1
  if d == 'en' then d = 'an' end
  local isdir = (d == 'r' or d == 'l')
  if outer == 'l' and d == 'an' then
    state.san = state.san or item
    state.ean = item
  elseif state.san then
    head, state = insert_numeric(head, state)
  end
  for r = first_on, q - 1 do
    nodes[r][2] = temp
    item = nodes[r][1]
    if Babel.mirroring_enabled and glyph_not_symbol_font(item)
      and temp == 'r' and characters[item.char] then
      local font_mode = ''
      if item.font > 0 and font.fonts[item.font].properties then
        font_mode = font.fonts[item.font].properties.mode
      end
      if font_mode ~= 'harf' and font_mode ~= 'plug' then
        item.char = characters[item.char].m or item.char
      end
    end
  end
  first_on = nil
end
if 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
  else
    if d == 'l' and state.sim and state.has_r then
      head, state = insert_implicit(head, state, outer)
    elseif d == 'l' then
      state.sim, state.eim, state.has_r = nil, nil, false
    end
  else
    if d == 'an' or d == 'l' then
      if nodes[q][3] then -- nil except after an explicit dir
        state.sim = item -- so we move sim 'inside' the group
      else
        state.sim = state.sim or item
      end
      state.eim = item
    elseif d == 'r' and state.sim then
      head, state = insert_implicit(head, state, outer)
    elseif d == 'r' then
      state.sim, state.eim = nil, nil
    end
  end
else
  if d == 'an' or d == 'l' then
    if nodes[q][3] then -- nil except after an explicit dir
      state.sim = item -- so we move sim 'inside' the group
    else
      state.sim = state.sim or item
    end
  end
  state.eim = item
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
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
11 Data for CJK

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

\[
\begin{align*}
[0x0021] &= \{c='ex'\}, \\
[0x0024] &= \{c='pr'\}, \\
[0x0025] &= \{c='po'\}, \\
[0x0028] &= \{c='op'\}, \\
[0x0029] &= \{c='cp'\}, \\
[0x002B] &= \{c='pr'\},
\end{align*}
\]

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

12 The 'nil' language

This 'language' does nothing, except setting the hyphenation patterns to nohyphenation. For this language currently no special definitions are needed or available.

The macro \LdfInit takes care of preventing that this file is loaded more than once, checking the category code of the @ sign, etc.

\begin{verbatim}
\ProvidesLanguage{nil}{}{(date) \langle\langle version\rangle\rangle Nil
language}
\LdfInit{nil}{datenil}
\end{verbatim}

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.

\begin{verbatim}
\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
\end{verbatim}

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

\begin{verbatim}
\providehyphenmins{\CurrentOption}{\m@ne\m@ne}
\end{verbatim}

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

\captionnil\datenil
\begin{verbatim}
\let\captionsnil\@empty
\let\datenil\@empty
\end{verbatim}

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

\begin{verbatim}
\def\bbl@inidata@nil{%
\bbl@elt{identification}{tag.ini}{und}%
\bbl@elt{identification}{load.level}{0}%
\bbl@elt{identification}{charset}{utf8}%
\bbl@elt{identification}{version}{1.0}%
\bbl@elt{identification}{date}{2022-05-16}%
\bbl@elt{identification}{name.local}{nil}%
\bbl@elt{identification}{name.english}{nil}%
\bbl@elt{identification}{name.babel}{nil}%
\bbl@elt{identification}{tag.bcp47}{und}%
\bbl@elt{identification}{language.tag.bcp47}{und}%
\end{verbatim}
13 Calendars

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

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

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

```
\ExplSyntaxOn
\def\bbl@ca@islamic#1-#2-#3\@@#4#5#6{}
\def\bbl@ca@islamicvl@x#1#2-#3-#4\@@#5#6#7{\if\bbl@cs@isltojd#1#2#3\@@#4#5#6\fi}
\@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
\@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
\@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{0}}
\@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
\@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
\ExplSyntaxOff
```

13.1 Islamic

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

```
\ExplSyntaxOn
\def\bbl@ca@islamiccivil#1-#2\@@#3#4#5#6{\bbl@ca@islamicvl@x{#1\#2\#3\@@#4#5#6}}
\def\bbl@ca@islamic-civil++\bbl@ca@islamiccivil{+2}
\def\bbl@ca@islamic-civil+\bbl@ca@islamiccivil{+1}
\def\bbl@ca@islamic-civil\bbl@ca@islamiccivil{0}
\def\bbl@ca@islamic-civil-\bbl@ca@islamiccivil{-1}
\def\bbl@ca@islamic-civil--\bbl@ca@islamiccivil{-2}
```

The Civil calendar.

```
\def\bbl@ca@islamiccivil#1-#2\@@#3#4#5#6{
\fp_eval:n{ (#3 + cell(29.5 * (#2 - 1)) + (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) + 1948439.5 - 1) }
\def\bbl@ca@islamic-civil++\bbl@ca@islamiccivil{+2}
\def\bbl@ca@islamic-civil+\bbl@ca@islamiccivil{+1}
\def\bbl@ca@islamic-civil\bbl@ca@islamiccivil{0}
\def\bbl@ca@islamic-civil-\bbl@ca@islamiccivil{-1}
\def\bbl@ca@islamic-civil--\bbl@ca@islamiccivil{-2}
```

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

Since the main aim is to provide a suitable \today, and maybe some close dates, data just covers Hijri ∼1435/∼1460 (Gregorian ∼2014/∼2038).

\def\bbl@cs@umalqura@data{56660, 56690, 56719, 56749, 56778, 56808, 56837, 56867, 56897, 56926, 56956, 56985, 57015, 57044, 57074, 57103, 57133, 57162, 57192, 57221, 57251, 57280, 57310, 57340, 57369, 57399, 57429, 57458, 57487, 57517, 57546, 57576, 57605, 57634, 57664, 57694, 57723, 57753, 57783, 57813, 57842, 57871, 57901, 57930, 57959, 57989, 58018, 58048, 58077, 58107, 58137, 58167, 58196, 58226, 58255, 58285, 58314, 58343, 58373, 58402, 58432, 58461, 58491, 58521, 58551, 58580, 58610, 58639, 58669, 58698, 58727, 58757, 58786, 58816, 58845, 58875, 58905, 58934, 58964, 58994, 59023, 59053, 59082, 59111, 59141, 59170, 59200, 59229, 59259, 59288, 59318, 59348, 59377, 59407, 59436, 59466, 59495, 59525, 59554, 59584, 59613, 59643, 59672, 59702, 59731, 59761, 59791, 59820, 59850, 59879, 59909, 59939, 59968, 59997, 60027, 60056, 60086, 60115, 60145, 60174, 60204, 60234, 60264, 60293, 60323, 60352, 60381, 60411, 60440, 60469, 60499, 60528, 60558, 60588, 60618, 60648, 60677, 60707, 60736, 60765, 60795, 60824, 60853, 60883, 60912, 60942, 60972, 61002, 61031, 61061, 61090, 61120, 61149, 61179, 61208, 61237, 61267, 61296, 61326, 61356, 61385, 61415, 61445, 61474, 61504, 61533, 61563, 61592, 61621, 61651, 61680, 61710, 61739, 61769, 61799, 61828, 61858, 61888, 61917, 61947, 61976, 62066, 62095, 62123, 62153, 62182, 62212, 62242, 62271, 62301, 62331, 62360, 62399, 62419, 62448, 62478, 62507, 62537, 62566, 62596, 62625, 62655, 62685, 62715, 62744, 62774, 62803, 62832, 62862, 62891, 62921, 62950, 62980, 63009, 63039, 63069, 63099, 63128, 63157, 63187, 63216, 63246, 63275, 63305, 63334, 63363, 63393, 63423, 63453, 63482, 63512, 63541, 63571, 63600, 63630, 63659, 63689, 63718, 63747, 63777, 63807, 63836, 63866, 63895, 63925, 63955, 63984, 64014, 64043, 64073, 64102, 64131, 64161, 64190, 64220, 64249, 64279, 64309, 64339, 64368, 64398, 64427, 64457, 64486, 64515, 64545, 64574, 64603, 64633, 64663, 64692, 64722, 64752, 64782, 64811, 64841, 64870, 64899, 64929, 64958, 64987, 65017, 65047, 65076, 65106, 65136, 65166, 65195, 65225, 65254, 65283, 65313, 65342, 65371, 65401, 65431, 65460, 65490, 65520}
13.2 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.
\def\bbl@GregDaySPriormonths{\tmpd}{\tmpb}{\bbl@gregdayspriormonths{\tmpd}{\tmpb}{\tmpc}}% 
\bbl@checkleaphebryear{\tmpa}{\tmpb}{\tmpc}{\global\bbl@hebrleaptrue}{\global\bbl@hebrleapfalse}\def\bbl@GregDaySPrioryears{\tmpd}{\tmpb}{\bbl@gregdaysprioryears{\tmpd}{\tmpb}{\global\bbl@hebrleaptrue}{\global\bbl@hebrleapfalse}}% 
\def\bbl@GregDayS{\tmpa}{\tmpb}{\tmpc}{\global\bbl@hebrleaptrue}{\global\bbl@hebrleapfalse}
\multiply \tmpa by 13753
\advance \tmpa by 5604
\bbl@remainder{\tmpa}{25920}{\tmpc}\% \tmpc == ConjunctionParts
\divide \tmpa by 25920
\multiply \#2 by 29
\advance \tmpa by 1
\advance \#2 by \tmpa
\bbl@remainder{\#2}{7}{\tmpa}\%
\ifnum \tmpc < 19440
\ifnum \tmpc < 9924
\else
\ifnum \tmpa=2
\bbl@checkleaphebryear{#1}\% of a common year
\ifbbl@hebrleap
\else
\advance \#2 by 1
\fi
\fi
\fi
\else
\ifnum \tmpc < 16789
\else
\ifnum \tmpa=1
\advance \#1 by -1
\bbl@checkleaphebryear{#1}\% at the end of leap year
\ifbbl@hebrleap
\advance \#2 by 1
\fi
\fi
\fi
\else
\advance \#2 by 1
\fi
\fi
\fi
\else
\advance \#2 by 1
\fi
\bbl@remainder{\#2}{7}{\tmpa}\%
\ifnum \tmpa=0
\advance \#2 by 1
\else
\ifnum \tmpa=3
\advance \#2 by 1
\fi
\fi
\global\bbl@cntcommon=\#2 \relax
\#2=\bbl@cntcommon}
def\bbl@daysinhebryear#1#2{%
countdef\tmpf=14
#3=\ifcase #1 \relax
0 \or
0 \or
30 \or
59 \or
89 \or
118 \or
154
#4=\ifcase #2 \relax
0 \or
5 \or
29 \or
58 \or
89 \or
118 \or
154
\bbl@checkleaphebryear(#2)\%  
\ifbbl@hebrleap  
  \ifnum #1 > 6  
    \advance #3 by 30  
  \fi  
\fi  
\bbl@daysinhebryear(#2){\tmpf}\%  
\ifnum #1 > 3  
  \ifnum \tmpf=353  
    \advance #3 by -1  
  \fi  
  \ifnum \tmpf=383  
    \advance #3 by -1  
  \fi  
\fi  
\ifnum #1 > 2  
  \ifnum \tmpf=355  
    \advance #3 by 1  
  \fi  
  \ifnum \tmpf=385  
    \advance #3 by 1  
  \fi  
\fi  
\global\bbl@cntcommon=#3\relax  
#3=\bbl@cntcommon\}  
def\bbl@absfromhebr#1#2#3#4{  
{#4=#1\relax  
\bbl@hebrdayspriormonths{#2}{#3}{#1}  
\advance #4 by #1\relax  
\bbl@hebrelapseddays{#3}{#1}  
\advance #4 by #1\relax  
\advance #4 by -1373429  
\global\bbl@cntcommon=#4\relax  
#4=\bbl@cntcommon\}  
def\bbl@hebrfromgreg#1#2#3#4#5#6{  
{\countdef\tmpx= 17  
\countdef\tmpy= 18  
\countdef\tmpz= 19  
#6=#3\relax  
\global\advance #6 by 3761  
\bbl@absfromgreg(#1){#2}{#3}{#4}\%  
\tmpz=1 \tmpy=1  
\bbl@absfromhebr{\tmpz}{\tmpy}{#6}\{\tmpx}\%  
\ifnum \tmpx > #4\relax  
\global\advance #6 by -1  
\bbl@absfromhebr{\tmpz}{\tmpy}{#6}\{\tmpx}\%  
\fi  
\advance #4 by -\tmpx  
\advance #4 by 1  
#5=#4\relax  
\divide #5 by 30  
\loop  
\bbl@hebrdayspriormonths{#5}{#6}\{\tmpx}\%
13.3 Persian

There is an algorithm written in TeX by Jabri, Abolhassani, Pournader and Esfahbod, created for the first versions of the FarsiTeX system (no longer available), but the original license is GPL, so its use with 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).

13.4 Coptic and Ethiopic

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

That's very simple.

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

14.1 Not renaming hyphen.tex

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

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

People can have a file 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
\plain\-tex to achieve the desired effect, based on the babel package. If you load each of them with
\input\TeX, you will get a file called either bplain.fmt or blplain.fmt, which you can use as
replacements for plain.fmt and \plain\-fmt.

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

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

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).

\openin0 hyphen.cfg
\ifeof0
\let\a\input
Then \input is defined to forget about its argument and load hyphen.cfg instead. Once that's done the original meaning of \input can be restored and the definition of \a can be forgotten.

\def\input #1 {%
\let\input\a
\a hyphen.cfg
\let\a\undefined
}
\fi
\loadlocalcfg

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

(bplain)\a plain.tex
(bplain)\a lplain.tex

Finally we change the contents of \fmtname to indicate that this is not the plain format, but a format based on plain with the babel package preloaded.

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

14.2 Emulating some \LaTeX features

The file babel.def expects some definitions made in the \LaTeX style file. So, in Plain we must provide at least some predefined values as well some tools to set them (even if not all options are available). There are no package options, and therefore and alternative mechanism is provided. For the moment, only \babeloptionstrings and \babeloptionmath are provided, which can be defined before loading babel. \BabelModifiers can be set too (but not sure it works).

\def\@empty{}
\def\loadlocalcfg#1{%
\openin0#1.cfg
\ifeof0
\closein0
\immediate\write16{*************************************}
\immediate\write16{* Local config file #1.cfg used}
\immediate\write16{*}
\input #1.cfg\relax
\fi\@endofldf}
14.3 General tools

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

\begin{verbatim}
\long\def\@firstofone#1{#1}
\long\def\@firstoftwo#1#2{#1}
\long\def\@secondoftwo#1#2{#2}
\def\@nnil{\@nil}
\def\@gobbletwo#1#2{ }
\def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
\def\@star@or@long#1{\@ifstar{\let\l@ngrel@x\relax#1}}
\let\l@ngrel@x\relax
\def\@car#1#2\@nil{#1}
\def\@cdr#1#2\@nil{#2}
\let\@typeset@protect\relax
\let\protected@edef\edef
\long\def\@gobble#1{ }
\edef\@backslashchar{\expandafter\@gobble\string\\}
\def\strip@prefix#1>{
\def\g@addto@macro#1#2{\toks@\expandafter{#1#2}\xdef#1{\the	oks@}}
\def\@namedef#1{\expandafter\def\csname #1\endcsname}
\def\@nameuse#1{\csname #1\endcsname}
\def\@ifundefined#1{\expandafter\ifx\csname#1\endcsname\relax\expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
\def\@expandtwoargs#1#2#3{\edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
\def\zap@space#1 #2{#1\ifx#2\@empty\else\expandafter\zap@space\fi #2}
\let\bbl@trace\@gobble
\def\bbl@error#1{\begingroup\catcode`\=0 \catcode`\%=12 \catcode`\^^M=5 \catcode`\%=14 \input errbabel.def \bbl@error{#1}}
\def\bbl@warning#1{\begingroup\newlinechar=`
\def\\{\^^J(babel) \message{\#1} \endgroup}
\let\bbl@infowarn\bbl@warning
\def\bbl@info#1{\begingroup\newlinechar=`
\def\\{\^^J \wlog{#1} \endgroup}
\end{verbatim}

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

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

\def\begindocument{\@begindocumenthook
\global\let\@begindocumenthook\@undefined
\def\do##1{\global\let##1\@undefined}%
\@preamblecmds
\global\let\do\noexpand}
\ifx\@begindocumenthook\@undefined
\def\@begindocumenthook{}
\fi
\@onlypreamble\@begindocumenthook
\def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

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

\def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
\@onlypreamble\AtEndOfPackage
\def\@endofldf{}
\@onlypreamble\@endofldf
\let\bbl@afterlang\@empty
\chardef\bbl@opt@hyphenmap\z@    
\LaTeX{} needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default. There is a trick to hide some conditional commands from the outer `\iff'. The same trick is applied below.

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

Mimic \LaTeX{}'s commands to define control sequences.

\def\newcommand{\@star@or@long\new@command}
\def\new@command#1{\@testopt{\@newcommand#1}0}
\def\@newcommand#1[#2]{\@ifnextchar [\@xargdef#1[#2]}{\@argdef#1[#2]}
\long\def\@argdef#1[#2]#3{\@yargdef#1\@ne{#2}{#3}}
\long\def\@xargdef#1[#2][#3]#4{\expandafter\def\expandafter#1\expandafter{\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}}\reserved@a}\@tempcnta\relax\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}}\reserved@a\@tempcnta\relax\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}}\reserved@a
\@tempcnta\relax\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}}\reserved@a
\@tempcnta\relax\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}}\reserved@a
The following little macro \in@ is taken from latex.ltx; it checks whether its first argument is part of its second argument. It uses the boolean \in@; allocating a new boolean inside conditionally executed code is not possible, hence the construct with the temporary definition of \bbl@tempa.

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

\catcode`\&=\z@ % Trick to hide conditionals
\def@x@protect#1&fi#2#3{&fi\protect#1}

The following little macro \in@ is taken from latex.ltx; it checks whether its first argument is part of its second argument. It uses the boolean \in@; allocating a new boolean inside conditionally executed code is not possible, hence the construct with the temporary definition of \bbl@tempa.

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

\catcode`\&=\z@ % Trick to hide conditionals
\def@x@protect#1&fi#2#3{&fi\protect#1}

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

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

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

\def\@ifpackagewith\#1\#2\#3\#4(#3)

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

\def\@ifl@aded\#1\#2\#3\#4{}}
For the following code we need to make sure that the commands \texttt{\newcommand} and \texttt{\providecommand} exist with some sensible definition. They are not fully equivalent to their \LaTeX\ versions; just enough to make things work in plain \TeX\ environments.

\begin{verbatim}
\@tempcnta\undefined
\csname newcount@endcsname\@tempcnta\relax
\@tempcntb\undefined
\csname newcount@endcsname\@tempcntb\relax
\fi
\ifx\@tempcntb\@undefined
\csname newcount@endcsname\@tempcntb\relax
\fi
\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=} 
\let\:{\@xifnch} \expandafter\def:\ {uturelet\@let@token\@ifnch}
\fi
\def\@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}}
\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{ 
\expandafter\enumerate#1}
\fi
\def\testopt#1{% 
\@testopt#1}
\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}}
\end{verbatim}

14.4 Encoding related macros

Code from \texttt{ltoutenc.dtx}, adapted for use in the plain \TeX\ environment.
Currently we only use the \HTeX{} method for accents for those that are known to be made active in some language definition file.

\begin{verbatim}
\DeclareTextAccent{"}{OT1}{127}
\DeclareTextAccent{'}{OT1}{19}
\DeclareTextAccent{^}{OT1}{94}
\DeclareTextAccent{`}{OT1}{18}
\DeclareTextAccent{~}{OT1}{126}
\end{verbatim}

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

\begin{verbatim}
\DeclareTextSymbol{"}{OT1}{92}
\DeclareTextSymbol{'}{OT1}{93}
\DeclareTextSymbol{^}{OT1}{94}
\DeclareTextSymbol{`}{OT1}{102}
\DeclareTextSymbol{\i}{OT1}{16}
\DeclareTextSymbol{\ss}{OT1}{25}
\end{verbatim}

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

\begin{verbatim}
\ifx\scriptsize\undefined
  \let\scriptsize\sevenrm
\end{verbatim}
And a few more “dummy” definitions.

–

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

⟨⟨ Emulate LaTeX ⟩⟩

A proxy file:

+plain
\input babel.def
//plain

15 Acknowledgements

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

References

