Babel

Code

Version 24.8.59765
2024/08/19

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Localization and internationalization

Unicode
TEX
pdfTEX
LuaTEX
XeTEX

Contents

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

1 Identification and loading of required files

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

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

There some additional `tex`, `def` and `lua` files
The babel installer extends docstrip with a few "pseudo-guards" to set "variables" used at installation time. They are used with `<@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

```
⟨⟨version=24.8.59765⟩⟩
⟨⟨date=2024/08/19⟩⟩
```

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.

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

```
\def\bbl@loop#1#2#3{\bbl@@loop#1{#3}#2,\@nnil,}
\def\bbl@loopx#1#2{\expandafter\bbl@loop\expandafter#1\expandafter{#2}}
\def\bbl@@loop#1#2#3,{%
  \ifx\@nnil#3\relax\else
    \def#1{#3}#2\bbl@afterfi\bbl@@loop#1{#2}%
  \fi}
\def\bbl@for#1#2#3{\bbl@loopx#1{#2}{\ifx#1\@empty\else#3\fi}}
```

`\bbl@add@list` This internal macro adds its second argument to a comma separated list in its first argument. When the list is not defined yet (or empty), it will be initiated. It presumes expandable character strings.

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

`\bbl@afterelse` `\bbl@afterfi` 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[1]. These macros will break if another `\if...\fi` statement appears in one of the arguments and it is not enclosed in braces.

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

`\bbl@exp` Now, just syntactical sugar, but it makes partial expansion of some code a lot more simple and readable. Here `\\` 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{%
  \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@trim@def`. The first one strips the leading and trailing spaces from the second argument and then applies the first argument (a macro, `\toks@` and the like). The second one, as its name suggests, defines the first argument as the stripped second argument.

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

`\bbl@ifunset` To check if a macro is defined, we create a new macro, which does the same as `\@ifundefined`. However, in an ϵ-tex engine, it is based on `\ifcsname`, which is more efficient, and does not waste

[1]This code is based on code presented in TUGboat vol. 12, no2, June 1991 in "An expansion Power Lemma" by Sonja Maus.

memory. Defined inside a group, to avoid `\ifcsname` being implicitly set to `\relax` by the `\csname` test.

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

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

```
\def\bbl@ifblank#1{%
  \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
\long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
\def\bbl@ifset#1#2#3{%
  \bbl@ifunset{#1}{#3}{\bbl@exp{\\\bbl@ifblank{\@nameuse{#1}}}{#3}{#2}}}
```

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

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

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

```
\def\bbl@vforeach#1#2{%
  \def\bbl@forcmd##1{#2}%
  \bbl@fornext#1,\@nil,}
\def\bbl@fornext#1,{%
  \ifx\@nil#1\relax\else
    \bbl@ifblank{#1}{}{\bbl@trim\bbl@forcmd{#1}}%
    \expandafter\bbl@fornext
  \fi}
\def\bbl@foreach#1{\expandafter\bbl@vforeach\expandafter{#1}}
```

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

```
\def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
  \toks@{}%
  \def\bbl@replace@aux##1#2##2#2{%
```

```
    \ifx\bbl@nil##2%
      \toks@\expandafter{\the\toks@##1}%
    \else
      \toks@\expandafter{\the\toks@##1#3}%
      \bbl@afterfi
      \bbl@replace@aux##2#2%
    \fi}%
  \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
  \edef#1{\the\toks@}}
```

An extension to the previous macro. It takes into account the parameters, and it is string based (ie, if you replace `elax` 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).

```
\ifx\detokenize\@undefined\else % Unused macros if old Plain TeX
  \bbl@exp{\def\\\bbl@parsedef##1\detokenize{macro:}}#2->#3\relax{%
    \def\bbl@tempa{#1}%
    \def\bbl@tempb{#2}%
    \def\bbl@tempe{#3}}
  \def\bbl@sreplace#1#2#3{%
    \begingroup
      \expandafter\bbl@parsedef\meaning#1\relax
      \def\bbl@tempc{#2}%
      \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
      \def\bbl@tempd{#3}%
      \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
      \bbl@xin@{\bbl@tempc}{\bbl@tempe}% If not in macro, do nothing
      \ifin@
        \bbl@exp{\\\bbl@replace\\\bbl@tempe{\bbl@tempc}{\bbl@tempd}}%
        \def\bbl@tempc{%     Expanded an executed below as 'uplevel'
           \\\makeatletter % "internal" macros with @ are assumed
           \\\scantokens{%
             \bbl@tempa\\\@namedef{\bbl@stripslash#1}\bbl@tempb{\bbl@tempe}}%
           \catcode64=\the\catcode64\relax}%  Restore @
      \else
        \let\bbl@tempc\@empty  % Not \relax
      \fi
      \bbl@exp{%      For the 'uplevel' assignments
    \endgroup
      \bbl@tempc}}  % empty or expand to set #1 with changes
\fi
```

Two further tools. `\bbl@ifsamestring` first expand its arguments and then compare their expansion (sanitized, so that the catcodes do not matter). `\bbl@engine` takes the following values: 0 is pdfTEX, 1 is luatex, and 2 is xetex. You may use the latter it in your language style if you want.

```
\def\bbl@ifsamestring#1#2{%
  \begingroup
    \protected@edef\bbl@tempb{#1}%
    \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
    \protected@edef\bbl@tempc{#2}%
    \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
    \ifx\bbl@tempb\bbl@tempc
      \aftergroup\@firstoftwo
    \else
      \aftergroup\@secondoftwo
    \fi
  \endgroup}
\chardef\bbl@engine=%
  \ifx\directlua\@undefined
    \ifx\XeTeXinputencoding\@undefined
      \z@
    \else
      \tw@
    \fi
  \else
    \@ne
  \fi
```

A somewhat hackish tool (hence its name) to avoid spurious spaces in some contexts.

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

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

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

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

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

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

```
⟨⟨*Make sure ProvidesFile is defined⟩⟩ ≡
\ifx\ProvidesFile\@undefined
  \def\ProvidesFile#1[#2 #3 #4]{%
    \wlog{File: #1 #4 #3 <#2>}%
    \let\ProvidesFile\@undefined}
\fi
⟨⟨/Make sure ProvidesFile is defined⟩⟩
```

3.1 Multiple languages

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

```
⟨⟨*Define core switching macros⟩⟩ ≡
```

```
\ifx\language\@undefined
  \csname newcount\endcsname\language
\fi
⟨⟨/Define core switching macros⟩⟩
```

`\last@language` Another counter is used to keep track of the allocated languages. TEX and LATEX reserves for this purpose the count 19.

`\addlanguage` This macro was introduced for TEX < 2. Preserved for compatibility.

```
⟨⟨*Define core switching macros⟩⟩ ≡
\countdef\last@language=19
\def\addlanguage{\csname newlanguage\endcsname}
⟨⟨/Define core switching macros⟩⟩
```

Now we make sure all required files are loaded. When the command `\AtBeginDocument` doesn't exist we assume that we are dealing with a plain-based format. In that case the file `plain.def` is needed (which also defines `\AtBeginDocument`, and therefore it is not loaded twice). We need the first part when the format is created, and `\orig@dump` is used as a flag. Otherwise, we need to use the second part, so `\orig@dump` is not defined (`plain.def` undefines it).
Check if the current version of `switch.def` has been previously loaded (mainly, `hyphen.cfg`). If not, load it now. We cannot load `babel.def` here because we first need to declare and process the package options.

3.2 The Package File (LATEX, `babel.sty`)

```
⟨*package⟩
\NeedsTeXFormat{LaTeX2e}[2005/12/01]
\ProvidesPackage{babel}[⟨⟨date⟩⟩ v⟨⟨version⟩⟩ The Babel package]
```

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

```
\@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}
  {\providecommand\bbl@trace[1]{}%
   \let\bbl@debug\@gobble
   \ifx\directlua\@undefined\else
     \directlua{ Babel = Babel or {}
       Babel.debug = false }%
   \fi}
\def\bbl@error#1{% Implicit #2#3#4
  \begingroup
    \catcode`\\=0 \catcode`\==12 \catcode`\`=12
    \input errbabel.def
  \endgroup
  \bbl@error{#1}}
\def\bbl@warning#1{%
  \begingroup
    \def\\{\MessageBreak}%
    \PackageWarning{babel}{#1}%
  \endgroup}
\def\bbl@infowarn#1{%
  \begingroup
    \def\\{\MessageBreak}%
    \PackageNote{babel}{#1}%
  \endgroup}
\def\bbl@info#1{%
  \begingroup
    \def\\{\MessageBreak}%
    \PackageInfo{babel}{#1}%
  \endgroup}
```

This file also takes care of a number of compatibility issues with other packages an defines a few additional package options. Apart from all the language options below we also have a few options that influence the behavior of language definition files.
Many of the following options don't do anything themselves, they are just defined in order to make it possible for babel and language definition files to check if one of them was specified by the user.
But first, include here the *Basic macros* defined above.

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

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

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

3.3 base

The first 'real' option to be processed is `base`, which set the hyphenation patterns then resets `ver@babel.sty` so that LaTeXforgets about the first loading. After a subset of `babel.def` has been loaded (the old `switch.def`) and `\AfterBabelLanguage` defined, it exits.
Now the `base` option. With it we can define (and load, with luatex) hyphenation patterns, even if we are not interested in the rest of babel.

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

3.4 key=value options and other general option

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

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

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

```
\DeclareOption{KeepShorthandsActive}{}
\DeclareOption{activeacute}{}
\DeclareOption{activegrave}{}
\DeclareOption{debug}{}
\DeclareOption{noconfigs}{}
\DeclareOption{showlanguages}{}
\DeclareOption{silent}{}
% \DeclareOption{mono}{}
\DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
\chardef\bbl@iniflag\z@
\DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne}    % main -> +1
\DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@}   % add = 2
\DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % add + main
% A separate option
\let\bbl@autoload@options\@empty
\DeclareOption{provide@=*}{\def\bbl@autoload@options{import}}
% Don't use. Experimental. TODO.
\newif\ifbbl@single
\DeclareOption{selectors=off}{\bbl@singletrue}
```

```
⟨⟨More package options⟩⟩
```

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

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

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

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

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

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

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

```
\ProcessOptions*
```

```
\ifx\bbl@opt@provide\@nnil
  \let\bbl@opt@provide\@empty  % %%% MOVE above
\else
  \chardef\bbl@iniflag\@ne
  \bbl@exp{\\\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
    \in@{,provide,}{,#1,}%
    \ifin@
      \def\bbl@opt@provide{#2}%
      \bbl@replace\bbl@opt@provide{;}{,}%
    \fi}
\fi
%
```

3.5 Conditional loading of shorthands

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

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

```
\ifx\bbl@opt@shorthands\@nnil
  \def\bbl@ifshorthand#1#2#3{#2}%
\else\ifx\bbl@opt@shorthands\@empty
  \def\bbl@ifshorthand#1#2#3{#3}%
\else
```

The following macro tests if a shorthand is one of the allowed ones.

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

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

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

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

```
  \bbl@ifshorthand{'}%
    {\PassOptionsToPackage{activeacute}{babel}}{}
  \bbl@ifshorthand{`}%
    {\PassOptionsToPackage{activegrave}{babel}}{}
\fi\fi
```

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

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

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

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

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

```
\bbl@trace{Defining IfBabelLayout}
\ifx\bbl@opt@layout\@nnil
  \newcommand\IfBabelLayout[3]{#3}%
\else
  \bbl@exp{\\\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
    \in@{,layout,}{,#1,}%
    \ifin@
      \def\bbl@opt@layout{#2}%
      \bbl@replace\bbl@opt@layout{ }{.}%
    \fi}
  \newcommand\IfBabelLayout[1]{%
    \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
    \ifin@
      \expandafter\@firstoftwo
    \else
      \expandafter\@secondoftwo
    \fi}
\fi
⟨/package⟩
⟨*core⟩
```

3.6 Interlude for Plain

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

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

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

```
⟨/core⟩
⟨*package | core⟩
```

4 Multiple languages

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

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

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

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

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

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

```
        {\edef\bbl@tempd{\def\noexpand#1{#1}}%
         \lowercase\expandafter{\bbl@tempd}}}%
      \@empty
    \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
  \bbl@tempd
  \bbl@exp{\\\bbl@usehooks{languagename}{{\languagename}{#1}}}}
\def\bbl@iflanguage#1{%
  \@ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}
```

After a name has been 'fixed', the selectors will try to load the language. If even the fixed name is not defined, will load it on the fly, either based on its name, or if activated, its BCP47 code.
We first need a couple of macros for a simple BCP 47 look up. It also makes sure, with `\bbl@bcpcase`, casing is the correct one, so that sr-latn-ba becomes fr-Latn-BA. Note #4 may contain some `\@empty`'s, but they are eventually removed. `\bbl@bcplookup` either returns the found `ini` or it is `\relax`.

```
\def\bbl@bcpcase#1#2#3#4\@@#5{%
  \ifx\@empty#3%
    \uppercase{\def#5{#1#2}}%
  \else
    \uppercase{\def#5{#1}}%
    \lowercase{\edef#5{#5#2#3#4}}%
  \fi}
\def\bbl@bcplookup#1-#2-#3-#4\@@{%
  \let\bbl@bcp\relax
  \lowercase{\def\bbl@tempa{#1}}%
  \ifx\@empty#2%
    \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
  \else\ifx\@empty#3%
    \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
    \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
      {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
      {}%
    \ifx\bbl@bcp\relax
      \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
    \fi
  \else
    \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
    \bbl@bcpcase#3\@empty\@empty\@@\bbl@tempc
    \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
      {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
      {}%
    \ifx\bbl@bcp\relax
      \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
        {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
        {}%
    \fi
    \ifx\bbl@bcp\relax
      \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
        {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
        {}%
    \fi
    \ifx\bbl@bcp\relax
      \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
    \fi
  \fi\fi}
\let\bbl@initoload\relax
⟨-core⟩
\def\bbl@provide@locale{%
  \ifx\babelprovide\@undefined
    \bbl@error{base-on-the-fly}{}{}{}%
  \fi
  \let\bbl@auxname\languagename % Still necessary. TODO
  \bbl@ifunset{bbl@bcp@map@\languagename}{}% Move uplevel??
    {\edef\languagename{\@nameuse{bbl@bcp@map@\languagename}}}%
  \ifbbl@bcpallowed
    \expandafter\ifx\csname date\languagename\endcsname\relax
      \expandafter
      \bbl@bcplookup\languagename-\@empty-\@empty-\@empty\@@
      \ifx\bbl@bcp\relax\else  % Returned by \bbl@bcplookup
        \edef\languagename{\bbl@bcp@prefix\bbl@bcp}%
        \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
        \expandafter\ifx\csname date\languagename\endcsname\relax
          \let\bbl@initoload\bbl@bcp
          \bbl@exp{\\\babelprovide[\bbl@autoload@bcpoptions]{\languagename}}%
          \let\bbl@initoload\relax
        \fi
        \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
      \fi
    \fi
  \fi
  \expandafter\ifx\csname date\languagename\endcsname\relax
    \IfFileExists{babel-\languagename.tex}%
      {\bbl@exp{\\\babelprovide[\bbl@autoload@options]{\languagename}}}%
      {}%
  \fi}
⟨+core⟩
```

`\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.

```
\def\iflanguage#1{%
  \bbl@iflanguage{#1}{%
    \ifnum\csname l@#1\endcsname=\language
      \expandafter\@firstoftwo
    \else
      \expandafter\@secondoftwo
    \fi}}
```

4.1 Selecting the language

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

```
\let\bbl@select@type\z@
\edef\selectlanguage{%
  \noexpand\protect
  \expandafter\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, arabi, koma). It is related to a trick for 2.09, now discarded.

```
\let\xstring\string
```

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

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

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

```
\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.

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

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

```
\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}%
  \let\bbl@ifrestoring\@secondoftwo}
```

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@last{0}    % No real need for a new counter
\def\bbl@id@assign{%
  \bbl@ifunset{bbl@id@@\languagename}%
    {\count@\bbl@id@last\relax
     \advance\count@\@ne
     \bbl@csarg\chardef{id@@\languagename}\count@
     \edef\bbl@id@last{\the\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}%
    {}%
    \chardef\localeid\bbl@cl{id@}}
```

The unprotected part of `\selectlanguage`. In case it is used as environment, declare `\endselectlaguage`, just for safety.

```
\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 `\languagename` are messed up. This is a bug, but preserved for backwards compatibility. The list of auxiliary files can be extended by redefining `\BabelContentsFiles`, but make sure they are loaded inside a group (as `aux`, `toc`, `lof`, and `lot` do) or the last language of the document will remain active afterwards.
We also write a command to change the current language in the auxiliary files.
`\bbl@savelastskip` is used to deal with skips before the write whatsit (as suggested by U Fischer). Adapted from hyperref, but it might fail, so I'll consider it a temporary hack, while I study other options (the ideal, but very likely unfeasible except perhaps in luatex, is to avoid the `\write` altogether when not needed).

```
\def\BabelContentsFiles{toc,lof,lot}
\def\bbl@set@language#1{% from selectlanguage, pop@
  % The old buggy way. Preserved for compatibility.
  \edef\languagename{%
    \ifnum\escapechar=\expandafter`\string#1\@empty
    \else\string#1\@empty\fi}%
  \ifcat\relax\noexpand#1%
    \expandafter\ifx\csname date\languagename\endcsname\relax
      \edef\languagename{#1}%
      \let\localename\languagename
    \else
      \bbl@info{Using '\string\language' instead of 'language' is\\%
                deprecated. If what you want is to use a\\%
                macro containing the actual locale, make\\%
                sure it does not not match any language.\\%
                Reported}%
      \ifx\scantokens\@undefined
         \def\localename{??}%
      \else
        \scantokens\expandafter{\expandafter
          \def\expandafter\localename\expandafter{\languagename}}%
      \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
    \if@filesw
      \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
        \bbl@savelastskip
        \protected@write\@auxout{}{\string\babel@aux{\bbl@auxname}{}}%
        \bbl@restorelastskip
      \fi
      \bbl@usehooks{write}{}%
    \fi
  \fi}
%
\let\bbl@restorelastskip\relax
\let\bbl@savelastskip\relax
%
\newif\ifbbl@bcpallowed
\bbl@bcpallowedfalse
\def\select@language#1{% from set@, babel@aux
  \ifx\bbl@selectorname\@empty
    \def\bbl@selectorname{select}%
  % set hymap
  \fi
  \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
  % set name
  \edef\languagename{#1}%
  \bbl@fixname\languagename
  % TODO. name@map must be here?
  \bbl@provide@locale
  \bbl@iflanguage\languagename{%
    \let\bbl@select@type\z@
    \expandafter\bbl@switch\expandafter{\languagename}}}
\def\babel@aux#1#2{%
  \select@language{#1}%
  \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
    \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}% TODO - plain?
\def\babel@toc#1#2{%
  \select@language{#1}}
```

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 *re*define `\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 `\`⟨*lang*⟩`hyphenmins` is defined. If it is not, we set default values (2 and 3), otherwise the values in `\`⟨*lang*⟩`hyphenmins` will be used.
No text is supposed to be added with switching captions and date, so we remove any spurious spaces with `\bbl@bsphack` and `\bbl@esphack`.

```
\newif\ifbbl@usedategroup
\let\bbl@savedextras\@empty
\def\bbl@switch#1{%  from select@, foreign@
  % make sure there is info for the language if so requested
  \bbl@ensureinfo{#1}%
  % restore
  \originalTeX
  \expandafter\def\expandafter\originalTeX\expandafter{%
    \csname noextras#1\endcsname
    \let\originalTeX\@empty
    \babel@beginsave}%
  \bbl@usehooks{afterreset}{}%
  \languageshorthands{none}%
  % set the locale id
  \bbl@id@assign
  % switch captions, date
  \bbl@bsphack
    \ifcase\bbl@select@type
      \csname captions#1\endcsname\relax
      \csname date#1\endcsname\relax
    \else
      \bbl@xin@{,captions,}{,\bbl@select@opts,}%
      \ifin@
        \csname captions#1\endcsname\relax
      \fi
      \bbl@xin@{,date,}{,\bbl@select@opts,}%
      \ifin@  % if \foreign... within \<lang>date
        \csname date#1\endcsname\relax
      \fi
    \fi
  \bbl@esphack
  % switch extras
  \csname bbl@preextras@#1\endcsname
  \bbl@usehooks{beforeextras}{}%
  \csname extras#1\endcsname\relax
  \bbl@usehooks{afterextras}{}%
  %  > babel-ensure
  %  > babel-sh-<short>
  %  > babel-bidi
  %  > babel-fontspec
  \let\bbl@savedextras\@empty
  % hyphenation - case mapping
  \ifcase\bbl@opt@hyphenmap\or
    \def\BabelLower##1##2{\lccode##1=##2\relax}%
    \ifnum\bbl@hymapsel>4\else
      \csname\languagename @bbl@hyphenmap\endcsname
    \fi
    \chardef\bbl@opt@hyphenmap\z@
  \else
    \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
      \csname\languagename @bbl@hyphenmap\endcsname
    \fi
  \fi
  \let\bbl@hymapsel\@cclv
  % hyphenation - select rules
  \ifnum\csname l@\languagename\endcsname=\l@unhyphenated
    \edef\bbl@tempa{u}%
  \else
    \edef\bbl@tempa{\bbl@cl{lnbrk}}%
  \fi
  % linebreaking - handle u, e, k (v in the future)
  \bbl@xin@{/u}{/\bbl@tempa}%
  \ifin@\else\bbl@xin@{/e}{/\bbl@tempa}\fi % elongated forms
  \ifin@\else\bbl@xin@{/k}{/\bbl@tempa}\fi % only kashida
  \ifin@\else\bbl@xin@{/p}{/\bbl@tempa}\fi % padding (eg, Tibetan)
  \ifin@\else\bbl@xin@{/v}{/\bbl@tempa}\fi % variable font
  \ifin@
    % unhyphenated/kashida/elongated/padding = allow stretching
    \language\l@unhyphenated
    \babel@savevariable\emergencystretch
    \emergencystretch\maxdimen
    \babel@savevariable\hbadness
    \hbadness\@M
  \else
    % other = select patterns
    \bbl@patterns{#1}%
  \fi
  % hyphenation - mins
  \babel@savevariable\lefthyphenmin
  \babel@savevariable\righthyphenmin
  \expandafter\ifx\csname #1hyphenmins\endcsname\relax
    \set@hyphenmins\tw@\thr@@\relax
  \else
    \expandafter\expandafter\expandafter\set@hyphenmins
```

```
      \csname #1hyphenmins\endcsname\relax
  \fi
  % reset selector name
  \let\bbl@selectorname\@empty}
```

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

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

`\foreignlanguage` The `\foreignlanguage` command is another substitute for the `\selectlanguage` command. This command takes two arguments, the first argument is the name of the language to use for typesetting the text specified in the second argument.
Unlike `\selectlanguage` this command doesn’t switch *everything*, it only switches the hyphenation rules and the extra definitions for the language specified. It does this within a group and assumes the `\extras`⟨*lang*⟩ command doesn’t make any `\global` changes. The coding is very similar to part of `\selectlanguage`.
`\bbl@beforeforeign` is a trick to fix a bug in bidi texts. `\foreignlanguage` is supposed to be a ‘text’ command, and therefore it must emit a `\leavevmode`, but it does not, and therefore the indent is placed on the opposite margin. For backward compatibility, however, it is done only if a right-to-left script is requested; otherwise, it is no-op.
(3.11) `\foreignlanguage*` is a temporary, experimental macro for a few lines with a different script direction, while preserving the paragraph format (thank the braces around `\par`, things like `\hangindent` are not reset). Do not use it in production, because its semantics and its syntax may change (and very likely will, or even it could be removed altogether). Currently it enters in vmode and then selects the language (which in turn sets the paragraph direction).
(3.11) Also experimental are the hook `foreign` and `foreign*`. With them you can redefine `\BabelText` which by default does nothing. Its behavior is not well defined yet. So, use it in horizontal mode only if you do not want surprises.
In other words, at the beginning of a paragraph `\foreignlanguage` enters into hmode with the surrounding lang, and with `\foreignlanguage*` with the new lang.

```
\providecommand\bbl@beforeforeign{}
\edef\foreignlanguage{%
  \noexpand\protect
  \expandafter\noexpand\csname foreignlanguage \endcsname}
\expandafter\def\csname foreignlanguage \endcsname{%
  \@ifstar\bbl@foreign@s\bbl@foreign@x}
\providecommand\bbl@foreign@x[3][]{%
  \begingroup
    \def\bbl@selectorname{foreign}%
    \def\bbl@select@opts{#1}%
    \let\BabelText\@firstofone
    \bbl@beforeforeign
    \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}
\providecommand\BabelWrapText[1]{%
   \def\bbl@tempa{\def\BabelText####1}%
   \expandafter\bbl@tempa\expandafter{\BabelText{#1}}}
```

`\foreign@language` This macro does the work for `\foreignlanguage` and the `otherlanguage*` 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
  \bbl@fixname\languagename
  % TODO. name@map here?
  \bbl@provide@locale
  \bbl@iflanguage\languagename{%
    \let\bbl@select@type\@ne
    \expandafter\bbl@switch\expandafter{\languagename}}}
```

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

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

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

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

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

```
\def\hyphenrules#1{%
  \edef\bbl@tempf{#1}%
  \bbl@fixname\bbl@tempf
  \bbl@iflanguage\bbl@tempf{%
    \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
    \ifx\languageshorthands\@undefined\else
      \languageshorthands{none}%
    \fi
    \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
      \set@hyphenmins\tw@\thr@@\relax
    \else
      \expandafter\expandafter\expandafter\set@hyphenmins
      \csname\bbl@tempf hyphenmins\endcsname\relax
    \fi}}
\let\endhyphenrules\@empty
```

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

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

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

```
\def\set@hyphenmins#1#2{%
  \lefthyphenmin#1\relax
  \righthyphenmin#2\relax}
```

`\ProvidesLanguage` The identification code for each file is something that was introduced in LaTeX 2_ε. When the command `\ProvidesFile` does not exist, a dummy definition is provided temporarily. For use in the language definition file the command `\ProvidesLanguage` is defined by babel.
Depending on the format, ie, on if the former is defined, we use a similar definition or not.

```
\ifx\ProvidesFile\@undefined
```

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

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

```
\ifx\originalTeX\@undefined\let\originalTeX\@empty\fi
```

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

```
\ifx\babel@beginsave\@undefined\let\babel@beginsave\relax\fi
```

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

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

4.2 Errors

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

\@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 LaTeX 2$_\varepsilon$, 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'. Please,\\%
    define it after the language has been loaded\\%
    (typically in the preamble) with:\\%
    \string\setlocalecaption{\languagename}{\bbl@tempa}{..}\\%
    Feel free to contribute on github.com/latex3/babel.\\%
    Reported}}
\def\bbl@tentative{\protect\bbl@tentative@i}
\def\bbl@tentative@i#1{%
  \bbl@warning{%
```

```
    Some functions for '#1' are tentative.\\%
    They might not work as expected and their behavior\\%
    could change in the future.\\%
    Reported}}
\def\@nolanerr#1{\bbl@error{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
```

Here ended the now discarded `switch.def`. Here also (currently) ends the base option.

```
\ifx\directlua\@undefined\else
  \ifx\bbl@luapatterns\@undefined
    \input luababel.def
  \fi
\fi
\bbl@trace{Compatibility with language.def}
\ifx\bbl@languages\@undefined
  \ifx\directlua\@undefined
    \openin1 = language.def % TODO. Remove hardcoded number
    \ifeof1
      \closein1
      \message{I couldn't find the file language.def}
    \else
      \closein1
      \begingroup
        \def\addlanguage#1#2#3#4#5{%
          \expandafter\ifx\csname lang@#1\endcsname\relax\else
            \global\expandafter\let\csname l@#1\expandafter\endcsname
              \csname lang@#1\endcsname
          \fi}%
        \def\uselanguage#1{}%
        \input language.def
      \endgroup
    \fi
  \fi
  \chardef\l@english\z@
\fi
```

`\addto` It takes two arguments, a ⟨*control sequence*⟩ and TEX-code to be added to the ⟨*control sequence*⟩. If the ⟨*control sequence*⟩ has not been defined before it is defined now. The control sequence could also expand to `\relax`, in which case a circular definition results. The net result is a stack overflow. Note there is an inconsistency, because the assignment in the last branch is global.

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

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

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

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

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

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

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

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

```
\def\bbl@redefinerobust#1{%
  \edef\bbl@tempa{\bbl@stripslash#1}%
  \bbl@ifunset{\bbl@tempa\space}%
    {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
     \bbl@exp{\def\\#1{\\\protect\<\bbl@tempa\space>}}}%
    {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}%
    \@namedef{\bbl@tempa\space}}
\@onlypreamble\bbl@redefinerobust
```

4.3 Hooks

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

```
\bbl@trace{Hooks}
\newcommand\AddBabelHook[3][]{%
  \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
  \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
  \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
  \bbl@ifunset{bbl@ev@#2@#3@#1}%
    {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
    {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
  \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
\newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
\newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
\def\bbl@usehooks{\bbl@usehooks@lang\languagename}
\def\bbl@usehooks@lang#1#2#3{% Test for Plain
  \ifx\UseHook\@undefined\else\UseHook{babel/*/#2}\fi
  \def\bbl@elth##1{%
    \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@}#3}}%
  \bbl@cs{ev@#2@}%
  \ifx\languagename\@undefined\else % Test required for Plain (?)
    \ifx\UseHook\@undefined\else\UseHook{babel/#1/#2}\fi
    \def\bbl@elth##1{%
      \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#1}#3}}%
    \bbl@cs{ev@#2@#1}%
  \fi}
```

To ensure forward compatibility, arguments in hooks are set implicitly. So, if a further argument is added in the future, there is no need to change the existing code. Note events intended for hyphen.cfg are also loaded (just in case you need them for some reason).

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

`\babelensure` The user command just parses the optional argument and creates a new macro named `\bbl@e@`⟨*language*⟩. We register a hook at the `afterextras` event which just executes this macro in a "complete" selection (which, if undefined, is `\relax` and does nothing). This part is somewhat involved because we have to make sure things are expanded the correct number of times.
The macro `\bbl@e@`⟨*language*⟩ contains `\bbl@ensure{`⟨*include*⟩`}{`⟨*exclude*⟩`}{`⟨*fontenc*⟩`}`, which in in turn loops over the macros names in `\bbl@captionslist`, excluding (with the help of `\in@`) those in the `exclude` list. If the `fontenc` is given (and not `\relax`), the `\fontencoding` is also added. Then we loop over the `include` list, but if the macro already contains `\foreignlanguage`, nothing is done. Note this macro (1) is not restricted to the preamble, and (2) changes are local.

```
\bbl@trace{Defining babelensure}
\newcommand\babelensure[2][]{%
  \AddBabelHook{babel-ensure}{afterextras}{%
    \ifcase\bbl@select@type
      \bbl@cl{e}%
    \fi}%
  \begingroup
    \let\bbl@ens@include\@empty
    \let\bbl@ens@exclude\@empty
    \def\bbl@ens@fontenc{\relax}%
    \def\bbl@tempb##1{%
      \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
    \edef\bbl@tempa{\bbl@tempb#1\@empty}%
    \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ens@##1}{##2}}%
    \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
    \def\bbl@tempc{\bbl@ensure}%
    \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
      \expandafter{\bbl@ens@include}}%
    \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
      \expandafter{\bbl@ens@exclude}}%
    \toks@\expandafter{\bbl@tempc}%
    \bbl@exp{%
  \endgroup
  \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}}
\def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
  \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
    \ifx##1\@undefined % 3.32 - Don't assume the macro exists
      \edef##1{\noexpand\bbl@nocaption
        {\bbl@stripslash##1}{\languagename\bbl@stripslash##1}}%
    \fi
    \ifx##1\@empty\else
      \in@{##1}{#2}%
      \ifin@\else
        \bbl@ifunset{bbl@ensure@\languagename}%
          {\bbl@exp{%
            \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
              \\\foreignlanguage{\languagename}%
              {\ifx\relax#3\else
                \\\fontencoding{#3}\\\selectfont
               \fi
               ########1}}}}%
          {}%
        \toks@\expandafter{##1}%
        \edef##1{%
           \bbl@csarg\noexpand{ensure@\languagename}%
           {\the\toks@}}%
      \fi
      \expandafter\bbl@tempb
    \fi}%
  \expandafter\bbl@tempb\bbl@captionslist\today\@empty
  \def\bbl@tempa##1{% elt for include list
    \ifx##1\@empty\else
      \bbl@csarg\in@{ensure@\languagename\expandafter}\expandafter{##1}%
      \ifin@\else
        \bbl@tempb##1\@empty
      \fi
      \expandafter\bbl@tempa
    \fi}%
  \bbl@tempa#1\@empty}
\def\bbl@captionslist{%
  \prefacename\refname\abstractname\bibname\chaptername\appendixname
  \contentsname\listfigurename\listtablename\indexname\figurename
  \tablename\partname\enclname\ccname\headtoname\pagename\seename
  \alsoname\proofname\glossaryname}
```

4.4 Setting up language files

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

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

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

Now we check whether we should perhaps stop the processing of this file. To do this we first need to check whether the second argument that is passed to `\LdfInit` is a control sequence. We do that by looking at the first token after passing #2 through `string`. When it is equal to `\@backslashchar` we are dealing with a control sequence which we can compare with `\@undefined`.

If so, we call `\ldf@quit` to set the main language, restore the category code of the @-sign and call `\endinput`

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

Finally we check `\originalTeX`.

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

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

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

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

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

```
\def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
  \bbl@afterlang
  \let\bbl@afterlang\relax
  \let\BabelModifiers\relax
  \let\bbl@screset\relax}%
\def\ldf@finish#1{%
  \loadlocalcfg{#1}%
  \bbl@afterldf{#1}%
  \expandafter\main@language\expandafter{#1}%
  \catcode`\@=\atcatcode \let\atcatcode\relax
  \catcode`\==\eqcatcode \let\eqcatcode\relax}
```

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.

```
\@onlypreamble\LdfInit
\@onlypreamble\ldf@quit
\@onlypreamble\ldf@finish
```

`\main@language` `\bbl@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.

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

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

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

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

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

4.5 Shorthands

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

```
\bbl@trace{Shorhands}
\def\bbl@add@special#1{% 1:a macro like \", \?, etc.
  \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
  \bbl@ifunset{@sanitize}{}{\bbl@add\@sanitize{\@makeother#1}}%
  \ifx\nfss@catcodes\@undefined\else % TODO - same for above
    \begingroup
      \catcode`#1\active
      \nfss@catcodes
      \ifnum\catcode`#1=\active
        \endgroup
        \bbl@add\nfss@catcodes{\@makeother#1}%
      \else
        \endgroup
      \fi
  \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 `\normal@char`⟨*char*⟩ to expand to the character in its 'normal state' and it defines the active character to expand to `\normal@char`⟨*char*⟩ by default (⟨*char*⟩ being the character to be made active). Later its definition can be changed to expand to `\active@char`⟨*char*⟩ by calling `\bbl@activate{`⟨*char*⟩`}`.
For example, to make the double quote character active one could have `\initiate@active@char{"}` in a language definition file. This defines " as `\active@prefix "\active@char"` (where the first " is the character with its original catcode, when the shorthand is created, and `\active@char"` is a single token). In protected contexts, it expands to `\protect "` or `\noexpand "` (ie, with the original "); otherwise `\active@char"` is executed. This macro in turn expands to `\normal@char"` in "safe" contexts (eg, `\label`), but `\user@active"` in normal "unsafe" ones. The latter search a definition in the user, language and system levels, in this order, but if none is found, `\normal@char"` is used. However, a deactivated shorthand (with `\bbl@deactivate` is defined as `\active@prefix "\normal@char"`.
The following macro is used to define shorthands in the three levels. It takes 4 arguments: the (string'ed) character, `\<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\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
    \else
      \bbl@afterfi\csname#2@sh@#1@\endcsname
    \fi}%
```

When there is also no current-level shorthand with an argument we will check whether there is a next-level defined shorthand for this active character.

```
  \long\@namedef{#3@arg#1}##1{%
    \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
      \bbl@afterelse\csname#4#1\endcsname##1%
    \else
      \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
    \fi}}%
```

`\initiate@active@char` calls `\@initiate@active@char` with 3 arguments. All of them are the same character with different catcodes: active, other (`\string`'ed) and the original one. This trick simplifies the code a lot.

```
\def\initiate@active@char#1{%
  \bbl@ifunset{active@char\string#1}%
    {\bbl@withactive
      {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
    {}}
```

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

```
\def\@initiate@active@char#1#2#3{%
  \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
  \ifx#1\@undefined
```

```
    \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
  \else
    \bbl@csarg\let{oridef@@#2}#1%
    \bbl@csarg\edef{oridef@#2}{%
      \let\noexpand#1%
      \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
  \fi
```

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

```
  \ifx#1#3\relax
    \expandafter\let\csname normal@char#2\endcsname#3%
  \else
    \bbl@info{Making #2 an active character}%
    \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
      \@namedef{normal@char#2}{%
        \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
    \else
      \@namedef{normal@char#2}{#3}%
    \fi
```

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

```
    \bbl@restoreactive{#2}%
    \AtBeginDocument{%
      \catcode`#2\active
      \if@filesw
        \immediate\write\@mainaux{\catcode`\string#2\active}%
      \fi}%
    \expandafter\bbl@add@special\csname#2\endcsname
    \catcode`#2\active
  \fi
```

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

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

```
    \expandafter\noexpand\csname user@active#2\endcsname}%
```

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

\active@prefix ⟨*char*⟩ \normal@char⟨*char*⟩

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

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

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

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

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

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

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

```
  \if\string'#2%
    \let\prim@s\bbl@prim@s
    \let\active@math@prime#1%
  \fi
  \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}}
```

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

```
⟨⟨*More package options⟩⟩ ≡
\DeclareOption{math=active}{}
\DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
⟨⟨/More package options⟩⟩
```

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

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

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

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

`\active@prefix` The command `\active@prefix` which is used in the expansion of active characters has a function similar to `\OT1-cmd` in that it `\protect`s 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.

```
\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}}
  {\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
```

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

```
\newif\if@safe@actives
\@safe@activesfalse
```

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

```
\def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}
```

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

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

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

```
\def\bbl@firstcs#1#2{\csname#1\endcsname}
\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.

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

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

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

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

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

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

```
\def\useshorthands{%
  \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}}
\def\bbl@usesh@s#1{%
  \bbl@usesh@x
    {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
    {#1}}
\def\bbl@usesh@x#1#2{%
  \bbl@ifshorthand{#2}%
    {\def\user@group{user}%
     \initiate@active@char{#2}%
     #1%
     \bbl@activate{#2}}%
    {\bbl@error{shorthand-is-off}{}{#2}{}}}
```

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

```
\def\user@language@group{user@\language@group}
\def\bbl@set@user@generic#1#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}%
      \@expandtwoargs
        \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
    \fi
    \declare@shorthand{\bbl@tempb}{#2}{#3}}}
```

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

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

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

```
\def\aliasshorthand#1#2{%
  \bbl@ifshorthand{#2}%
    {\expandafter\ifx\csname active@char\string#2\endcsname\relax
       \ifx\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}{}{#2}{}}}
```

\@notshorthand

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

\shorthandon \shorthandoff 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}%
      {\bbl@error{not-a-shorthand-b}{}{#2}{}}%
      {\ifcase#1%   off, on, off*
         \catcode`#212\relax
       \or
         \catcode`#2\active
         \bbl@ifunset{bbl@shdef@\string#2}%
           {}%
           {\bbl@withactive{\expandafter\let\expandafter}#2%
              \csname bbl@shdef@\string#2\endcsname
            \bbl@csarg\let{shdef@\string#2}\relax}%
         \ifcase\bbl@activated\or
           \bbl@activate{#2}%
         \else
           \bbl@deactivate{#2}%
         \fi
       \or
         \bbl@ifunset{bbl@shdef@\string#2}%
           {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
           {}%
         \csname bbl@oricat@\string#2\endcsname
         \csname bbl@oridef@\string#2\endcsname
       \fi}%
    \bbl@afterfi\bbl@switch@sh#1%
  \fi}
```

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}
  \let\bbl@s@activate\bbl@activate
  \def\bbl@activate#1{%
    \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
  \let\bbl@s@deactivate\bbl@deactivate
  \def\bbl@deactivate#1{%
    \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
\fi
```

You may want to test if a character is a shorthand. Note it does not test whether the shorthand is on or off.

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

`\bbl@prim@s` `\bbl@pr@m@s` One of the internal macros that are involved in substituting `\prime` for each right quote in mathmode 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`\*=\active  \lccode`\*=`\^
  \catcode`\'=12 \catcode`\"=\active  \lccode`\"=`\'
  \lowercase{%
    \gdef\bbl@pr@m@s{%
      \bbl@if@primes"'%
        \pr@@@s
        {\bbl@if@primes*^\pr@@@t\egroup}}}
\endgroup
```

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

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

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

```
\expandafter\def\csname OT1dqpos\endcsname{127}
\expandafter\def\csname T1dqpos\endcsname{4}
```

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

```
\ifx\f@encoding\@undefined
  \def\f@encoding{OT1}
\fi
```

4.6 Language attributes

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

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

```
\bbl@trace{Language attributes}
\newcommand\languageattribute[2]{%
  \def\bbl@tempc{#1}%
  \bbl@fixname\bbl@tempc
  \bbl@iflanguage\bbl@tempc{%
    \bbl@vforeach{#2}{%
```

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

```
      \ifx\bbl@known@attribs\@undefined
        \in@false
      \else
        \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
      \fi
      \ifin@
        \bbl@warning{%
          You have more than once selected the attribute '##1'\\%
          for language #1. Reported}%
      \else
```

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

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

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

```
\newcommand*{\@attrerr}[2]{%
  \bbl@error{unknown-attribute}{#1}{#2}{}}
```

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

```
\def\bbl@declare@ttribute#1#2#3{%
  \bbl@xin@{,#2,}{,\BabelModifiers,}%
  \ifin@
    \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
  \fi
  \bbl@add@list\bbl@attributes{#1-#2}%
  \expandafter\def\csname#1@attr@#2\endcsname{#3}}
```

`\bbl@ifattributeset` This internal macro has 4 arguments. It can be used to interpret TEX code based on whether a certain attribute was set. This command should appear inside the argument to `\AtBeginDocument` because the attributes are set in the document preamble, *after* 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.

```
\def\bbl@ifattributeset#1#2#3#4{%
  \ifx\bbl@known@attribs\@undefined
    \in@false
  \else
    \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
  \fi
  \ifin@
    \bbl@afterelse#3%
  \else
    \bbl@afterfi#4%
  \fi}
```

`\bbl@ifknown@ttrib` An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the TEX-code to be executed when the attribute is known and the TEX-code to be executed otherwise.

We first assume the attribute is unknown. Then we loop over the list of known attributes, trying to find a match.

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

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

```
\def\bbl@clear@ttribs{%
  \ifx\bbl@attributes\@undefined\else
    \bbl@loopx\bbl@tempa{\bbl@attributes}{%
      \expandafter\bbl@clear@ttrib\bbl@tempa.}%
    \let\bbl@attributes\@undefined
  \fi}
\def\bbl@clear@ttrib#1-#2.{%
  \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
\AtBeginDocument{\bbl@clear@ttribs}
```

4.7 Support for saving macro definitions

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

\babel@savecnt
\babel@beginsave

The initialization of a new save cycle: reset the counter to zero.

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

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

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

\babel@save
\babel@savevariable

The macro `\babel@save`⟨*csname*⟩ saves the current meaning of the control sequence ⟨*csname*⟩ to `\originalTeX`[2]. To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to `\originalTeX` and the counter is incremented. The macro `\babel@savevariable`⟨*variable*⟩ saves the value of the variable. ⟨*variable*⟩ can be anything allowed after the `\the` primitive. To avoid messing saved definitions up, they are saved only the very first time.

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

\bbl@frenchspacing
\bbl@nonfrenchspacing

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

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

[2] `\originalTeX` has to be expandable, i. e. you shouldn't let it to `\relax`.

```
      \sfcode`##1=##3\relax
    \fi}%
  \bbl@fs@chars
\else\if y\bbl@tempa     % french
  \def\bbl@elt##1##2##3{%
    \ifnum\sfcode`##1=##3\relax
      \babel@savevariable{\sfcode`##1}%
      \sfcode`##1=##2\relax
    \fi}%
  \bbl@fs@chars
\fi\fi\fi}
```

4.8 Short tags

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

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

4.9 Hyphens

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

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

\bbl@allowhyphens This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak \hskip 0pt plus 0pt`[3].

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

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

```
\newcommand\babelnullhyphen{\char\hyphenchar\font}
\def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
\def\bbl@hyphen{%
  \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
\def\bbl@hyphen@i#1#2{%
  \bbl@ifunset{bbl@hy@#1#2\@empty}%
    {\csname bbl@#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{%
  \ifnum\hyphenchar\font=\m@ne
    \babelnullhyphen
  \else
    \char\hyphenchar\font
  \fi}
```

Finally, we define the hyphen “types”. Their names will not change, so you may use them in `ldf`’s. After a space, the `\mbox` in `\bbl@hy@nobreak` is redundant.

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

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

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

[3] TEX begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

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`.

```
⟨⟨*More package options⟩⟩ ≡
\DeclareOption{nocase}{}
⟨⟨/More package options⟩⟩
```

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

```
⟨⟨*More package options⟩⟩ ≡
\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}
⟨⟨/More package options⟩⟩
```

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
  ⟨⟨Macros local to BabelCommands⟩⟩
  \def\bbl@provstring##1##2{%
    \providecommand##1{##2}%
    \bbl@toglobal##1}%
  \global\let\bbl@scafter\@empty
  \let\StartBabelCommands\bbl@startcmds
  \ifx\BabelLanguages\relax
     \let\BabelLanguages\CurrentOption
  \fi
  \begingroup
  \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
  \StartBabelCommands}
\def\bbl@startcmds{%
  \ifx\bbl@screset\@nnil\else
    \bbl@usehooks{stopcommands}{}%
  \fi
  \endgroup
  \begingroup
  \@ifstar
    {\ifx\bbl@opt@strings\@nnil
       \let\bbl@opt@strings\BabelStringsDefault
     \fi
     \bbl@startcmds@i}%
    \bbl@startcmds@i}
\def\bbl@startcmds@i#1#2{%
  \edef\bbl@L{\zap@space#1 \@empty}%
  \edef\bbl@G{\zap@space#2 \@empty}%
  \bbl@startcmds@ii}
\let\bbl@startcommands\StartBabelCommands
```

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

```
\newcommand\bbl@startcmds@ii[1][\@empty]{%
  \let\SetString\@gobbletwo
  \let\bbl@stringdef\@gobbletwo
  \let\AfterBabelCommands\@gobble
  \ifx\@empty#1%
    \def\bbl@sc@label{generic}%
    \def\bbl@encstring##1##2{%
      \ProvideTextCommandDefault##1{##2}%
      \bbl@toglobal##1%
      \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
    \let\bbl@sctest\in@true
  \else
    \let\bbl@sc@charset\space % <- zapped below
    \let\bbl@sc@fontenc\space % <-   "      "
    \def\bbl@tempa##1=##2\@nil{%
      \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
    \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
    \def\bbl@tempa##1 ##2{% space -> comma
      ##1%
      \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
    \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
    \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
    \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
    \def\bbl@encstring##1##2{%
      \bbl@foreach\bbl@sc@fontenc{%
        \bbl@ifunset{T@####1}%
          {}%
          {\ProvideTextCommand##1{####1}{##2}%
           \bbl@toglobal##1%
           \expandafter
           \bbl@toglobal\csname####1\string##1\endcsname}}}%
    \def\bbl@sctest{%
      \bbl@xin@{,\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
  \fi
  \ifx\bbl@opt@strings\@nnil         % ie, no strings key -> defaults
  \else\ifx\bbl@opt@strings\relax    % ie, strings=encoded
    \let\AfterBabelCommands\bbl@aftercmds
    \let\SetString\bbl@setstring
    \let\bbl@stringdef\bbl@encstring
  \else       % ie, strings=value
  \bbl@sctest
  \ifin@
    \let\AfterBabelCommands\bbl@aftercmds
    \let\SetString\bbl@setstring
    \let\bbl@stringdef\bbl@provstring
  \fi\fi\fi
  \bbl@scswitch
  \ifx\bbl@G\@empty
    \def\SetString##1##2{%
      \bbl@error{missing-group}{##1}{}{}}%
  \fi
  \ifx\@empty#1%
    \bbl@usehooks{defaultcommands}{}%
  \else
    \@expandtwoargs
    \bbl@usehooks{encodedcommands}{{\bbl@sc@charset}{\bbl@sc@fontenc}}%
  \fi}
```

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

```
\def\bbl@forlang#1#2{%
  \bbl@for#1\bbl@L{%
    \bbl@xin@{,#1,}{,\BabelLanguages,}%
    \ifin@#2\relax\fi}}
\def\bbl@scswitch{%
  \bbl@forlang\bbl@tempa{%
    \ifx\bbl@G\@empty\else
      \ifx\SetString\@gobbletwo\else
        \edef\bbl@GL{\bbl@G\bbl@tempa}%
        \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
        \ifin@\else
          \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
          \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
        \fi
      \fi
    \fi}}
\AtEndOfPackage{%
  \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
  \let\bbl@scswitch\relax}
\@onlypreamble\EndBabelCommands
\def\EndBabelCommands{%
  \bbl@usehooks{stopcommands}{}%
  \endgroup
  \endgroup
  \bbl@scafter}
\let\bbl@endcommands\EndBabelCommands
```

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

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

```
\def\bbl@setstring#1#2{% eg, \prefacename{<string>}
  \bbl@forlang\bbl@tempa{%
    \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
    \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
      {\bbl@exp{%
         \global\\\bbl@add\<\bbl@G\bbl@tempa>{\\\bbl@scset\\#1\<\bbl@LC>}}}%
      {}%
    \def\BabelString{#2}%
    \bbl@usehooks{stringprocess}{}%
    \expandafter\bbl@stringdef
      \csname\bbl@LC\expandafter\endcsname\expandafter{\BabelString}}}
```

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

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

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

```
⟨⟨*Macros local to BabelCommands⟩⟩ ≡
\def\SetStringLoop##1##2{%
  \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
  \count@\z@
  \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
    \advance\count@\@ne
    \toks@\expandafter{\bbl@tempa}%
    \bbl@exp{%
      \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
      \count@=\the\count@\relax}}}%
⟨⟨/Macros local to BabelCommands⟩⟩
```

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.

```
⟨⟨*Macros local to BabelCommands⟩⟩ ≡
  \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 local to BabelCommands⟩⟩
```

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

```
⟨⟨*Macros local to BabelCommands⟩⟩ ≡
  \newcommand\SetHyphenMap[1]{%
    \bbl@forlang\bbl@tempa{%
      \expandafter\bbl@stringdef
        \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
⟨⟨/Macros local to BabelCommands⟩⟩
```

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

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

The following package options control the behavior of hyphenation mapping.

```
⟨⟨*More package options⟩⟩ ≡
\DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
\DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
\DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
\DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
\DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
⟨⟨/More package options⟩⟩
```

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

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

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

```
\newcommand\setlocalecaption{%  TODO. Catch typos.
  \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
\def\bbl@setcaption@x#1#2#3{%  language caption-name string
  \bbl@trim@def\bbl@tempa{#2}%
  \bbl@xin@{.template}{\bbl@tempa}%
  \ifin@
    \bbl@ini@captions@template{#3}{#1}%
  \else
    \edef\bbl@tempd{%
      \expandafter\expandafter\expandafter
      \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
    \bbl@xin@
      {\expandafter\string\csname #2name\endcsname}%
      {\bbl@tempd}%
    \ifin@ % Renew caption
      \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
      \ifin@
        \bbl@exp{%
          \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
            {\\\bbl@scset\<#2name>\<#1#2name>}%
            {}}%
      \else % Old way converts to new way
        \bbl@ifunset{#1#2name}%
          {\bbl@exp{%
            \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
            \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
              {\def\<#2name>{\<#1#2name>}}%
              {}}}%
          {}%
      \fi
    \else
```

```
      \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
      \ifin@ % New way
        \bbl@exp{%
          \\\bbl@add\<captions#1>{\\\bbl@scset\<#2name>\<#1#2name>}%
          \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
            {\\\bbl@scset\<#2name>\<#1#2name>}%
            {}}%
      \else  % Old way, but defined in the new way
        \bbl@exp{%
          \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
          \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
            {\def\<#2name>{\<#1#2name>}}%
            {}}%
      \fi%
    \fi
    \@namedef{#1#2name}{#3}%
    \toks@\expandafter{\bbl@captionslist}%
    \bbl@exp{\\\in@{\<#2name>}{\the\toks@}}%
    \ifin@\else
      \bbl@exp{\\\bbl@add\\\bbl@captionslist{\<#2name>}}%
      \bbl@toglobal\bbl@captionslist
    \fi
  \fi}
% \def\bbl@setcaption@s#1#2#3{} % TODO. Not yet implemented (w/o 'name')
```

4.11 Macros common to a number of languages

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

```
\bbl@trace{Macros related to glyphs}
\def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
    \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
    \setbox\z@\hbox{\lower\dimen\z@ \box\z@}\ht\z@\ht\tw@ \dp\z@\dp\tw@}
```

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

```
\def\save@sf@q#1{\leavevmode
  \begingroup
    \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
  \endgroup}
```

4.12 Making glyphs available

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

4.12.1 Quotation marks

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

```
\ProvideTextCommand{\quotedblbase}{OT1}{%
  \save@sf@q{\set@low@box{\textquotedblright\/}%
    \box\z@\kern-.04em\bbl@allowhyphens}}
```

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

```
\ProvideTextCommandDefault{\quotedblbase}{%
  \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
\ProvideTextCommand{\quotesinglbase}{OT1}{%
  \save@sf@q{\set@low@box{\textquoteright\/}%
    \box\z@\kern-.04em\bbl@allowhyphens}}
```

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

```
\ProvideTextCommandDefault{\quotesinglbase}{%
  \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
\ProvideTextCommand{\guillemetleft}{OT1}{%
  \ifmmode
    \ll
  \else
    \save@sf@q{\nobreak
      \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
  \fi}
\ProvideTextCommand{\guillemetright}{OT1}{%
  \ifmmode
    \gg
  \else
    \save@sf@q{\nobreak
      \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
  \fi}
\ProvideTextCommand{\guillemotleft}{OT1}{%
  \ifmmode
    \ll
  \else
    \save@sf@q{\nobreak
      \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
  \fi}
\ProvideTextCommand{\guillemotright}{OT1}{%
  \ifmmode
    \gg
  \else
    \save@sf@q{\nobreak
      \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
  \fi}
```

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

```
\ProvideTextCommandDefault{\guillemetleft}{%
  \UseTextSymbol{OT1}{\guillemetleft}}
\ProvideTextCommandDefault{\guillemetright}{%
  \UseTextSymbol{OT1}{\guillemetright}}
\ProvideTextCommandDefault{\guillemotleft}{%
  \UseTextSymbol{OT1}{\guillemotleft}}
\ProvideTextCommandDefault{\guillemotright}{%
  \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
\ProvideTextCommand{\guilsinglleft}{OT1}{%
  \ifmmode
    <%
  \else
    \save@sf@q{\nobreak
      \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
  \fi}
\ProvideTextCommand{\guilsinglright}{OT1}{%
  \ifmmode
    >%
  \else
    \save@sf@q{\nobreak
      \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
  \fi}
```

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

```
\ProvideTextCommandDefault{\guilsinglleft}{%
```

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

4.12.2 Letters

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

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

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

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

4.12.3 Shorthands for quotation marks

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

\glq
\grq
The 'german' single quotes.

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

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

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

\glqq
\grqq
The 'german' double quotes.

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

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

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

\flq
\frq
The 'french' single guillemets.

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

\flqq
\frqq
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}%
  \let\bbl@umlaute\bbl@umlauta}
\def\umlautlow{%
  \def\bbl@umlauta{\protect\lower@umlaut}}
\def\umlautelow{%
  \def\bbl@umlaute{\protect\lower@umlaut}}
\umlauthigh
```

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

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

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

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

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

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

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

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

4.13 Layout

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

```
\bbl@trace{Bidi layout}
\providecommand\IfBabelLayout[3]{#3}%
⟨-core⟩
\newcommand\BabelPatchSection[1]{%
  \@ifundefined{#1}{}{%
```

```
    \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
    \@namedef{#1}{%
      \@ifstar{\bbl@presec@s{#1}}%
              {\@dblarg{\bbl@presec@x{#1}}}}}}
\def\bbl@presec@x#1[#2]#3{%
  \bbl@exp{%
    \\\select@language@x{\bbl@main@language}%
    \\\bbl@cs{sspre@#1}%
    \\\bbl@cs{ss@#1}%
      [\\\foreignlanguage{\languagename}{\unexpanded{#2}}]%
      {\\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
    \\\select@language@x{\languagename}}}
\def\bbl@presec@s#1#2{%
  \bbl@exp{%
    \\\select@language@x{\bbl@main@language}%
    \\\bbl@cs{sspre@#1}%
    \\\bbl@cs{ss@#1}*%
      {\\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
    \\\select@language@x{\languagename}}}
\IfBabelLayout{sectioning}%
  {\BabelPatchSection{part}%
   \BabelPatchSection{chapter}%
   \BabelPatchSection{section}%
   \BabelPatchSection{subsection}%
   \BabelPatchSection{subsubsection}%
   \BabelPatchSection{paragraph}%
   \BabelPatchSection{subparagraph}%
   \def\babel@toc#1{%
     \select@language@x{\bbl@main@language}}}{}
\IfBabelLayout{captions}%
  {\BabelPatchSection{caption}}{}
⟨+core⟩
```

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.

```
\bbl@trace{Input engine specific macros}
\ifcase\bbl@engine
  \input txtbabel.def
\or
  \input luababel.def
\or
  \input xebabel.def
\fi
\providecommand\babelfont{\bbl@error{only-lua-xe}{}{}{}}
\providecommand\babelprehyphenation{\bbl@error{only-lua}{}{}{}}
\ifx\babelposthyphenation\@undefined
  \let\babelposthyphenation\babelprehyphenation
  \let\babelpatterns\babelprehyphenation
  \let\babelcharproperty\babelprehyphenation
\fi
```

4.15 Creating and modifying languages

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

```
⟨/package | core⟩
⟨∗package⟩
\bbl@trace{Creating languages and reading ini files}
```

```
\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}%
    {\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}\@ne\tw@}%
  % == init ==
  \ifx\bbl@screset\@undefined
    \bbl@ldfinit
  \fi
  % == date (as option) ==
  % \ifx\bbl@KVP@date\@nnil\else
  % \fi
  % ==
  \let\bbl@lbkflag\relax % \@empty = do setup linebreak, only in 3 cases:
  \ifcase\bbl@howloaded
    \let\bbl@lbkflag\@empty % new
  \else
    \ifx\bbl@KVP@hyphenrules\@nnil\else
       \let\bbl@lbkflag\@empty
    \fi
    \ifx\bbl@KVP@import\@nnil\else
      \let\bbl@lbkflag\@empty
    \fi
  \fi
  % == import, captions ==
  \ifx\bbl@KVP@import\@nnil\else
    \bbl@exp{\\\bbl@ifblank{\bbl@KVP@import}}%
      {\ifx\bbl@initoload\relax
         \begingroup
           \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
           \bbl@input@texini{#2}%
         \endgroup
       \else
         \xdef\bbl@KVP@import{\bbl@initoload}%
       \fi}%
      {}%
    \let\bbl@KVP@date\@empty
  \fi
  \let\bbl@KVP@captions@@\bbl@KVP@captions % TODO. A dirty hack
  \ifx\bbl@KVP@captions\@nnil
    \let\bbl@KVP@captions\bbl@KVP@import
  \fi
  % ==
  \ifx\bbl@KVP@transforms\@nnil\else
    \bbl@replace\bbl@KVP@transforms{ }{,}%
  \fi
  % == Load ini ==
  \ifcase\bbl@howloaded
    \bbl@provide@new{#2}%
  \else
    \bbl@ifblank{#1}%
      {}%  With \bbl@load@basic below
      {\bbl@provide@renew{#2}}%
  \fi
  % == include == TODO
  % \ifx\bbl@included@inis\@empty\else
  %   \bbl@replace\bbl@included@inis{ }{,}%
  %   \bbl@foreach\bbl@included@inis{%
  %     \openin\bbl@readstream=babel-##1.ini
  %     \bbl@extend@ini{#2}}%
  %   \closein\bbl@readstream
  % \fi
  % Post tasks
  % ----------
  % == subsequent calls after the first provide for a locale ==
  \ifx\bbl@inidata\@empty\else
    \bbl@extend@ini{#2}%
  \fi
  % == ensure captions ==
  \ifx\bbl@KVP@captions\@nnil\else
    \bbl@ifunset{bbl@extracaps@#2}%
      {\bbl@exp{\\\babelensure[exclude=\\\today]{#2}}}%
      {\bbl@exp{\\\babelensure[exclude=\\\today,
                include=\[bbl@extracaps@#2]}]{#2}}%
    \bbl@ifunset{bbl@ensure@\languagename}%
      {\bbl@exp{%
        \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
          \\\foreignlanguage{\languagename}%
          {####1}}}}%
      {}%
    \bbl@exp{%
       \\\bbl@toglobal\<bbl@ensure@\languagename>%
       \\\bbl@toglobal\<bbl@ensure@\languagename\space>}%
  \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.

```
\bbl@load@basic{#2}%
% == script, language ==
% Override the values from ini or defines them
\ifx\bbl@KVP@script\@nnil\else
  \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
\fi
\ifx\bbl@KVP@language\@nnil\else
  \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
\fi
\ifcase\bbl@engine\or
```

```
  \bbl@ifunset{bbl@chrng@\languagename}{}%
    {\directlua{
       Babel.set_chranges_b('\bbl@cl{sbcp}', '\bbl@cl{chrng}') }}%
\fi
 % == onchar ==
\ifx\bbl@KVP@onchar\@nnil\else
  \bbl@luahyphenate
  \bbl@exp{%
    \\\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@
    \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
    \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
    \directlua{
      if Babel.script_blocks['\bbl@cl{sbcp}'] then
        Babel.loc_to_scr[\the\localeid] =
          Babel.script_blocks['\bbl@cl{sbcp}']
      end}%
    \ifx\bbl@mapselect\@undefined  % TODO. almost the same as mapfont
      \AtBeginDocument{%
        \bbl@patchfont{{\bbl@mapselect}}%
        {\selectfont}}%
      \def\bbl@mapselect{%
        \let\bbl@mapselect\relax
        \edef\bbl@prefontid{\fontid\font}}%
      \def\bbl@mapdir##1{%
        \begingroup
          \setbox\z@\hbox{% Force text mode
            \def\languagename{##1}%
            \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
            \bbl@switchfont
            \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
              \directlua{
                Babel.locale_props[\the\csname bbl@id@@##1\endcsname]%
                        ['/\bbl@prefontid'] = \fontid\font\space}%
            \fi}%
        \endgroup}%
    \fi
    \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
  \fi
  % TODO - catch non-valid values
\fi
% == mapfont ==
% For bidi texts, to switch the font based on direction
\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##1{%
      {\def\languagename{##1}%
       \let\bbl@ifrestoring\@firstoftwo % avoid font warning
       \bbl@switchfont
       \directlua{Babel.fontmap
         [\the\csname bbl@wdir@##1\endcsname]%
         [\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
\ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
  \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
\fi
\bbl@provide@intraspace
% == Line breaking: CJK quotes == TODO -> @extras
\ifcase\bbl@engine\or
  \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
  \ifin@
    \bbl@ifunset{bbl@quote@\languagename}{}%
      {\directlua{
         Babel.locale_props[\the\localeid].cjk_quotes = {}
         local cs = 'op'
         for c in string.utfvalues(%
             [[\csname bbl@quote@\languagename\endcsname]]) do
           if Babel.cjk_characters[c].c == 'qu' then
             Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
           end
           cs = ( cs == 'op') and 'cl' or 'op'
         end
      }}%
  \fi
\fi
% == Line breaking: justification ==
\ifx\bbl@KVP@justification\@nnil\else
   \let\bbl@KVP@linebreaking\bbl@KVP@justification
\fi
\ifx\bbl@KVP@linebreaking\@nnil\else
  \bbl@xin@{,\bbl@KVP@linebreaking,}%
```

```
    {,elongated,kashida,cjk,padding,unhyphenated,}%
  \ifin@
    \bbl@csarg\xdef
      {lnbrk@\languagename}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
  \fi
\fi
\bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
\ifin@\else\bbl@xin@{/k}{/\bbl@cl{lnbrk}}\fi
\ifin@\bbl@arabicjust\fi
\bbl@xin@{/p}{/\bbl@cl{lnbrk}}%
\ifin@\AtBeginDocument{\@nameuse{bbl@tibetanjust}}\fi
% == Line breaking: hyphenate.other.(locale|script) ==
\ifx\bbl@lbkflag\@empty
  \bbl@ifunset{bbl@hyotl@\languagename}{}%
    {\bbl@csarg\bbl@replace{hyotl@\languagename}{ }{,}%
     \bbl@startcommands*{\languagename}{}%
       \bbl@csarg\bbl@foreach{hyotl@\languagename}{%
         \ifcase\bbl@engine
           \ifnum##1<257
             \SetHyphenMap{\BabelLower{##1}{##1}}%
           \fi
         \else
           \SetHyphenMap{\BabelLower{##1}{##1}}%
         \fi}%
     \bbl@endcommands}%
  \bbl@ifunset{bbl@hyots@\languagename}{}%
    {\bbl@csarg\bbl@replace{hyots@\languagename}{ }{,}%
     \bbl@csarg\bbl@foreach{hyots@\languagename}{%
       \ifcase\bbl@engine
         \ifnum##1<257
           \global\lccode##1=##1\relax
         \fi
       \else
         \global\lccode##1=##1\relax
       \fi}}%
\fi
% == Counters: maparabic ==
% Native digits, if provided in ini (TeX level, xe and lua)
\ifcase\bbl@engine\else
  \bbl@ifunset{bbl@dgnat@\languagename}{}%
    {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
      \expandafter\expandafter\expandafter
      \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname
      \ifx\bbl@KVP@maparabic\@nnil\else
        \ifx\bbl@latinarabic\@undefined
          \expandafter\let\expandafter\@arabic
            \csname bbl@counter@\languagename\endcsname
        \else    % ie, if layout=counters, which redefines \@arabic
          \expandafter\let\expandafter\bbl@latinarabic
            \csname bbl@counter@\languagename\endcsname
        \fi
      \fi
    \fi}%
\fi
% == Counters: mapdigits ==
% > luababel.def
% == Counters: alph, Alph ==
\ifx\bbl@KVP@alph\@nnil\else
  \bbl@exp{%
    \\\bbl@add\<bbl@preextras@\languagename>{%
      \\\babel@save\\\@alph
      \let\\\@alph\<bbl@cntr@\bbl@KVP@alph @\languagename>}}%
\fi
```

```
\ifx\bbl@KVP@Alph\@nnil\else
  \bbl@exp{%
    \\\bbl@add\<bbl@preextras@\languagename>{%
      \\\babel@save\\\@Alph
      \let\\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\languagename>}}%
\fi
% == Casing ==
\bbl@release@casing
\ifx\bbl@KVP@casing\@nnil\else
  \bbl@csarg\xdef{casing@\languagename}%
    {\@nameuse{bbl@casing@\languagename}\bbl@maybextx\bbl@KVP@casing}%
\fi
% == Calendars ==
\ifx\bbl@KVP@calendar\@nnil
  \edef\bbl@KVP@calendar{\bbl@cl{calpr}}%
\fi
\def\bbl@tempe##1 ##2\@@{% Get first calendar
  \def\bbl@tempa{##1}}%
  \bbl@exp{\\\bbl@tempe\bbl@KVP@calendar\space\\\@@}%
\def\bbl@tempe##1.##2.##3\@@{%
  \def\bbl@tempc{##1}%
  \def\bbl@tempb{##2}}%
\expandafter\bbl@tempe\bbl@tempa..\@@
\bbl@csarg\edef{calpr@\languagename}{%
  \ifx\bbl@tempc\@empty\else
    calendar=\bbl@tempc
  \fi
  \ifx\bbl@tempb\@empty\else
    ,variant=\bbl@tempb
  \fi}%
% == engine specific extensions ==
% Defined in XXXbabel.def
\bbl@provide@extra{#2}%
% == require.babel in ini ==
% To load or reaload the babel-*.tex, if require.babel in ini
\ifx\bbl@beforestart\relax\else  % But not in doc aux or body
  \bbl@ifunset{bbl@rqtex@\languagename}{}%
    {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
       \let\BabelBeforeIni\@gobbletwo
       \chardef\atcatcode=\catcode`\@
       \catcode`\@=11\relax
       \def\CurrentOption{#2}%
       \bbl@input@texini{\bbl@cs{rqtex@\languagename}}%
       \catcode`\@=\atcatcode
       \let\atcatcode\relax
       \global\bbl@csarg\let{rqtex@\languagename}\relax
     \fi}%
  \bbl@foreach\bbl@calendars{%
    \bbl@ifunset{bbl@ca@##1}{%
      \chardef\atcatcode=\catcode`\@
      \catcode`\@=11\relax
      \InputIfFileExists{babel-ca-##1.tex}{}{}%
      \catcode`\@=\atcatcode
      \let\atcatcode\relax}%
    {}}%
\fi
% == frenchspacing ==
\ifcase\bbl@howloaded\in@true\else\in@false\fi
\ifin@\else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
\ifin@
  \bbl@extras@wrap{\\\bbl@pre@fs}%
    {\bbl@pre@fs}%
    {\bbl@post@fs}%
\fi
% == transforms ==
% > luababel.def
\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}%
      \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\\\today{\\\bbl@nocaption{today}{#1today}}}%
    \else
      \bbl@savetoday
      \bbl@savedate
    \fi
  \bbl@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@hyphens{#1}%
  \ifx\bbl@KVP@main\@nnil\else
     \expandafter\main@language\expandafter{#1}%
  \fi}
%
\def\bbl@provide@renew#1{%
  \ifx\bbl@KVP@captions\@nnil\else
    \StartBabelCommands*{#1}{captions}%
      \bbl@read@ini{\bbl@KVP@captions}2%   % Here all letters cat = 11
    \EndBabelCommands
  \fi
  \ifx\bbl@KVP@date\@nnil\else
    \StartBabelCommands*{#1}{date}%
      \bbl@savetoday
      \bbl@savedate
    \EndBabelCommands
  \fi
  % == hyphenrules (also in new) ==
  \ifx\bbl@lbkflag\@empty
    \bbl@provide@hyphens{#1}%
  \fi}
```

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.)

```
\def\bbl@load@basic#1{%
  \ifcase\bbl@howloaded\or\or
    \ifcase\csname bbl@llevel@\languagename\endcsname
      \bbl@csarg\let{lname@\languagename}\relax
    \fi
  \fi
  \bbl@ifunset{bbl@lname@#1}%
    {\def\BabelBeforeIni##1##2{%
       \begingroup
         \let\bbl@ini@captions@aux\@gobbletwo
         \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6{}%
         \bbl@read@ini{##1}1%
         \ifx\bbl@initoload\relax\endinput\fi
       \endgroup}%
     \begingroup       % boxed, to avoid extra spaces:
       \ifx\bbl@initoload\relax
         \bbl@input@texini{#1}%
       \else
         \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
       \fi
     \endgroup}%
    {}}
```

The hyphenrules option is handled with an auxiliary macro. This macro is called in three cases: when a language is first declared with \babelprovide, with hyphenrules and with import.

```
\def\bbl@provide@hyphens#1{%
  \@tempcnta\m@ne  % a flag
  \ifx\bbl@KVP@hyphenrules\@nnil\else
    \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
    \bbl@foreach\bbl@KVP@hyphenrules{%
      \ifnum\@tempcnta=\m@ne   % if not yet found
        \bbl@ifsamestring{##1}{+}%
          {\bbl@carg\addlanguage{l@##1}}%
          {}%
        \bbl@ifunset{l@##1}% After a possible +
          {}%
          {\@tempcnta\@nameuse{l@##1}}%
      \fi}%
    \ifnum\@tempcnta=\m@ne
      \bbl@warning{%
        Requested 'hyphenrules' for '\languagename' not found:\\%
        \bbl@KVP@hyphenrules.\\%
        Using the default value. Reported}%
    \fi
  \fi
  \ifnum\@tempcnta=\m@ne          % if no opt or no language in opt found
```

```
    \ifx\bbl@KVP@captions@@\@nnil % TODO. Hackish. See above.
      \bbl@ifunset{bbl@hyphr@#1}{}% use value in ini, if exists
        {\bbl@exp{\\\bbl@ifblank{\bbl@cs{hyphr@#1}}}%
           {}%
           {\bbl@ifunset{l@\bbl@cl{hyphr}}%
             {}%                  if hyphenrules found:
             {\@tempcnta\@nameuse{l@\bbl@cl{hyphr}}}}}%
    \fi
  \fi
  \bbl@ifunset{l@#1}%
    {\ifnum\@tempcnta=\m@ne
       \bbl@carg\adddialect{l@#1}\language
     \else
       \bbl@carg\adddialect{l@#1}\@tempcnta
     \fi}%
    {\ifnum\@tempcnta=\m@ne\else
       \global\bbl@carg\chardef{l@#1}\@tempcnta
     \fi}}
```

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`\\\%=\the\catcode`\%\relax
      \catcode`\\\\=\the\catcode`\\\relax
      \catcode`\\\{=\the\catcode`\{\relax
      \catcode`\\\}=\the\catcode`\}\relax}%
  \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\@@{%      full (default)
  \bbl@trim@def\bbl@tempa{#1}%
  \bbl@trim\toks@{#2}%
  \bbl@xin@{;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
  \ifin@\else
    \bbl@xin@{,identification/include.}%
             {,\bbl@section/\bbl@tempa}%
    \ifin@\xdef\bbl@included@inis{\the\toks@}\fi
    \bbl@exp{%
      \\\g@addto@macro\\\bbl@inidata{%
        \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
  \fi}
\def\bbl@inistore@min#1=#2\@@{%  minimal (maybe set in \bbl@read@ini)
  \bbl@trim@def\bbl@tempa{#1}%
  \bbl@trim\toks@{#2}%
  \bbl@xin@{.identification.}{.\bbl@section.}%
  \ifin@
    \bbl@exp{\\\g@addto@macro\\\bbl@inidata{%
      \\\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
  \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 \babelprovide it's either 1 or 2.

```
\def\bbl@loop@ini{%
  \loop
    \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
      \endlinechar\m@ne
      \read\bbl@readstream to \bbl@line
      \endlinechar`\^^M
      \ifx\bbl@line\@empty\else
        \expandafter\bbl@iniline\bbl@line\bbl@iniline
      \fi
    \repeat}
\ifx\bbl@readstream\@undefined
  \csname newread\endcsname\bbl@readstream
\fi
\def\bbl@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\bbl@csarg\let{inidata@\languagename}\bbl@inidata
    \global\let\bbl@inidata\@empty
    \bbl@exp{\\\bbl@add@list\\\bbl@ini@loaded{\languagename}}%
    \bbl@toglobal\bbl@ini@loaded
  \fi
  \closein\bbl@readstream}
\def\bbl@read@ini@aux{%
  \let\bbl@savestrings\@empty
  \let\bbl@savetoday\@empty
  \let\bbl@savedate\@empty
  \def\bbl@elt##1##2##3{%
    \def\bbl@section{##1}%
    \in@{=date.}{=##1}% Find a better place
    \ifin@
      \bbl@ifunset{bbl@inikv@##1}%
        {\bbl@ini@calendar{##1}}%
        {}%
    \fi
    \bbl@ifunset{bbl@inikv@##1}{}%
      {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
  \bbl@inidata}
```

A variant to be used when the ini file has been already loaded, because it's not the first \babelprovide for this language.

```
\def\bbl@extend@ini@aux#1{%
  \bbl@startcommands*{#1}{captions}%
    % Activate captions/... and modify exports
    \bbl@csarg\def{inikv@captions.licr}##1##2{%
      \setlocalecaption{#1}{##1}{##2}}%
    \def\bbl@inikv@captions##1##2{%
      \bbl@ini@captions@aux{##1}{##2}}%
    \def\bbl@stringdef##1##2{\gdef##1{##2}}%
    \def\bbl@exportkey##1##2##3{%
      \bbl@ifunset{bbl@@kv@##2}{}%
        {\expandafter\ifx\csname bbl@@kv@##2\endcsname\@empty\else
           \bbl@exp{\global\let\<bbl@##1@\languagename>\<bbl@@kv@##2>}%
         \fi}}%
    % As with \bbl@read@ini, but with some changes
    \bbl@read@ini@aux
    \bbl@ini@exports\tw@
    % Update inidata@lang by pretending the ini is read.
    \def\bbl@elt##1##2##3{%
      \def\bbl@section{##1}%
      \bbl@iniline##2=##3\bbl@iniline}%
    \csname bbl@inidata@#1\endcsname
    \global\bbl@csarg\let{inidata@#1}\bbl@inidata
  \StartBabelCommands*{#1}{date}% And from the import stuff
    \def\bbl@stringdef##1##2{\gdef##1{##2}}%
    \bbl@savetoday
    \bbl@savedate
  \bbl@endcommands}
```

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

```
\def\bbl@ini@calendar#1{%
 \lowercase{\def\bbl@tempa{=#1=}}%
 \bbl@replace\bbl@tempa{=date.gregorian}{}%
 \bbl@replace\bbl@tempa{=date.}{}%
 \in@{.licr=}{#1=}%
 \ifin@
   \ifcase\bbl@engine
     \bbl@replace\bbl@tempa{.licr=}{}%
   \else
     \let\bbl@tempa\relax
   \fi
 \fi
 \ifx\bbl@tempa\relax\else
   \bbl@replace\bbl@tempa{=}{}%
   \ifx\bbl@tempa\@empty\else
     \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
   \fi
   \bbl@exp{%
     \def\<bbl@inikv@#1>####1####2{%
       \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
 \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).

```
\def\bbl@renewinikey#1/#2\@@#3{%
  \edef\bbl@tempa{\zap@space #1 \@empty}%   section
  \edef\bbl@tempb{\zap@space #2 \@empty}%   key
  \bbl@trim\toks@{#3}%                      value
  \bbl@exp{%
    \edef\\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
    \\\g@addto@macro\\\bbl@inidata{%
       \\\bbl@elt{\bbl@tempa}{\bbl@tempb}{\the\toks@}}}}%
```

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

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

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

```
\def\bbl@iniwarning#1{%
  \bbl@ifunset{bbl@@kv@identification.warning#1}{}%
    {\bbl@warning{%
       From babel-\bbl@cs{lini@\languagename}.ini:\\%
       \bbl@cs{@kv@identification.warning#1}\\%
       Reported }}}
%
\let\bbl@release@transforms\@empty
\let\bbl@release@casing\@empty
\def\bbl@ini@exports#1{%
  % Identification always exported
  \bbl@iniwarning{}%
  \ifcase\bbl@engine
    \bbl@iniwarning{.pdflatex}%
  \or
    \bbl@iniwarning{.lualatex}%
  \or
    \bbl@iniwarning{.xelatex}%
  \fi%
  \bbl@exportkey{llevel}{identification.load.level}{}%
  \bbl@exportkey{elname}{identification.name.english}{}%
  \bbl@exp{\\\bbl@exportkey{lname}{identification.name.opentype}%
    {\csname bbl@elname@\languagename\endcsname}}%
  \bbl@exportkey{tbcp}{identification.tag.bcp47}{}%
  % Somewhat hackish. TODO:
  \bbl@exportkey{casing}{identification.tag.bcp47}{}%
  \bbl@exportkey{lbcp}{identification.language.tag.bcp47}{}%
  \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
  \bbl@exportkey{esname}{identification.script.name}{}%
  \bbl@exp{\\\bbl@exportkey{sname}{identification.script.name.opentype}%
    {\csname bbl@esname@\languagename\endcsname}}%
  \bbl@exportkey{sbcp}{identification.script.tag.bcp47}{}%
  \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
  \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
  \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
  \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%
  \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
  \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
  % Also maps bcp47 -> languagename
  \ifbbl@bcptoname
    \bbl@csarg\xdef{bcp@map@\bbl@cl{tbcp}}{\languagename}%
  \fi
  \ifcase\bbl@engine\or
    \directlua{%
      Babel.locale_props[\the\bbl@cs{id@@\languagename}].script
        = '\bbl@cl{sbcp}'}%
  \fi
  % Conditional
  \ifnum#1>\z@         % 0 = only info, 1, 2 = basic, (re)new
    \bbl@exportkey{calpr}{date.calendar.preferred}{}%
    \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
    \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
    \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
    \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
    \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
    \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
    \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
    \bbl@exportkey{intsp}{typography.intraspace}{}%
    \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
    \bbl@exportkey{chrng}{characters.ranges}{}%
    \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
    \bbl@exportkey{dgnat}{numbers.digits.native}{}%
    \ifnum#1=\tw@           % only (re)new
      \bbl@exportkey{rqtex}{identification.require.babel}{}%
      \bbl@toglobal\bbl@savetoday
      \bbl@toglobal\bbl@savedate
      \bbl@savestrings
    \fi
  \fi}
```

A shared handler for key=val lines to be stored in `\bbl@@kv@<section>.<key>`.

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

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

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

```
\def\bbl@inikv@counters#1#2{%
  \bbl@ifsamestring{#1}{digits}%
    {\bbl@error{digits-is-reserved}{}{}{}}%
    {}%
  \def\bbl@tempc{#1}%
  \bbl@trim@def{\bbl@tempb*}{#2}%
  \in@{.1$}{#1$}%
  \ifin@
    \bbl@replace\bbl@tempc{.1}{}%
    \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\languagename}{%
      \noexpand\bbl@alphnumeral{\bbl@tempc}}%
  \fi
  \in@{.F.}{#1}%
  \ifin@\else\in@{.S.}{#1}\fi
  \ifin@
    \bbl@csarg\protected@xdef{cntr@#1@\languagename}{\bbl@tempb*}%
  \else
    \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
    \expandafter\bbl@buildifcase\bbl@tempb* \\ % Space after \\
    \bbl@csarg{\global\expandafter\let}{cntr@#1@\languagename}\bbl@tempa
  \fi}
```

Now captions and `captions.licr`, depending on the engine. And below also for dates. They rely on a few auxiliary macros. It is expected the ini file provides the complete set in Unicode and LICR, in that order.

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

The auxiliary macro for captions define `\<caption>name`.

```
\def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
  \bbl@replace\bbl@tempa{.template}{}%
  \def\bbl@toreplace{#1{}}%
  \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
  \bbl@replace\bbl@toreplace{[[}{\csname}%
  \bbl@replace\bbl@toreplace{[}{\csname the}%
  \bbl@replace\bbl@toreplace{]]}{name\endcsname{}}%
  \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@#2}\bbl@toreplace
  \fi
  \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
  \ifin@
    \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
    \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
      \\\bbl@ifunset{bbl@\bbl@tempa fmt@\\\languagename}%
        {\[fnum@\bbl@tempa]}%
        {\\\@nameuse{bbl@\bbl@tempa fmt@\\\languagename}}}}%
  \fi}
\def\bbl@ini@captions@aux#1#2{%
  \bbl@trim@def\bbl@tempa{#1}%
  \bbl@xin@{.template}{\bbl@tempa}%
  \ifin@
    \bbl@ini@captions@template{#2}\languagename
  \else
    \bbl@ifblank{#2}%
      {\bbl@exp{%
         \toks@{\\\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}}%
      {\bbl@trim\toks@{#2}}%
    \bbl@exp{%
      \\\bbl@add\\\bbl@savestrings{%
        \\\SetString\<\bbl@tempa name>{\the\toks@}}}%
    \toks@\expandafter{\bbl@captionslist}%
    \bbl@exp{\\\in@{\<\bbl@tempa name>}{\the\toks@}}%
    \ifin@\else
      \bbl@exp{%
        \\\bbl@add\<bbl@extracaps@\languagename>{\<\bbl@tempa name>}%
        \\\bbl@toglobal\<bbl@extracaps@\languagename>}%
    \fi
  \fi}
```

Labels. Captions must contain just strings, no format at all, so there is new group in ini files.

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

```
          \\\babel@save\<labelenum\romannumeral\bbl@tempa>%
          \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
        \\\bbl@toglobal\<extras\languagename>}%
    \fi
  \fi}
```

To show correctly some captions in a few languages, we need to patch some internal macros, because the order is hardcoded. For example, in Japanese the chapter number is surrounded by two 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}%
        {\@chapapp\space\thechapter}
        {\@nameuse{bbl@\bbl@chaptype fmt@\languagename}}}
    \bbl@add\appendix{\def\bbl@chaptype{appendix}}% Not harmful, I hope
    \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
    \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
    \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
    \bbl@toglobal\appendix
    \bbl@toglobal\ps@headings
    \bbl@toglobal\chaptermark
    \bbl@toglobal\@makechapterhead}
  \let\bbl@patchappendix\bbl@patchchapter
\fi\fi\fi
\ifx\@part\@undefined
  \let\bbl@patchpart\relax
\else
  \def\bbl@patchpart{%
    \global\let\bbl@patchpart\relax
    \gdef\bbl@partformat{%
      \bbl@ifunset{bbl@partfmt@\languagename}%
        {\partname\nobreakspace\thepart}
        {\@nameuse{bbl@partfmt@\languagename}}}
    \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
    \bbl@toglobal\@part}
\fi
```

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

```
\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@cl{calpr}},%
      #1}%
    \bbl@replace\bbl@tempe{ }{}%
    \bbl@replace\bbl@tempe{CONVERT}{convert=}% Hackish
    \bbl@replace\bbl@tempe{convert}{convert=}%
    \let\bbl@ld@calendar\@empty
    \let\bbl@ld@variant\@empty
    \let\bbl@ld@convert\relax
    \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ld@##1}{##2}}%
    \bbl@foreach\bbl@tempe{\bbl@tempb##1\@@}%
    \bbl@replace\bbl@ld@calendar{gregorian}{}%
    \ifx\bbl@ld@calendar\@empty\else
      \ifx\bbl@ld@convert\relax\else
        \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
          {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
      \fi
    \fi
    \@nameuse{bbl@precalendar}% Remove, eg, +, -civil (-ca-islamic)
    \edef\bbl@calendar{% Used in \month..., too
      \bbl@ld@calendar
      \ifx\bbl@ld@variant\@empty\else
        .\bbl@ld@variant
      \fi}%
    \bbl@cased
      {\@nameuse{bbl@date@\languagename @\bbl@calendar}%
         \bbl@they\bbl@them\bbl@thed}%
  \endgroup}
% eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
\def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
  \bbl@trim@def\bbl@tempa{#1.#2}%
  \bbl@ifsamestring{\bbl@tempa}{months.wide}%      to savedate
    {\bbl@trim@def\bbl@tempa{#3}%
     \bbl@trim\toks@{#5}%
     \@temptokena\expandafter{\bbl@savedate}%
     \bbl@exp{%   Reverse order - in ini last wins
       \def\\\bbl@savedate{%
         \\\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
         \the\@temptokena}}}%
    {\bbl@ifsamestring{\bbl@tempa}{date.long}%     defined now
      {\lowercase{\def\bbl@tempb{#6}}%
       \bbl@trim@def\bbl@toreplace{#5}%
       \bbl@TG@@date
       \global\bbl@csarg\let{date@\languagename @\bbl@tempb}\bbl@toreplace
       \ifx\bbl@savetoday\@empty
         \bbl@exp{% TODO. Move to a better place.
           \\\AfterBabelCommands{%
             \def\<\languagename date>{\\\protect\<\languagename date >}%
             \\\newcommand\<\languagename date >[4][]{%
               \\\bbl@usedategrouptrue
               \<bbl@ensure@\languagename>{%
                 \\\localedate[####1]{####2}{####3}{####4}}}}%
           \def\\\bbl@savetoday{%
             \\\SetString\\\today{%
               \<\languagename date>[convert]%
                  {\\\the\year}{\\\the\month}{\\\the\day}}}}%
       \fi}%
      {}}}
```

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

```
\let\bbl@calendar\@empty
\newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
  \@nameuse{bbl@ca@#2}#1\@@}
\newcommand\BabelDateSpace{\nobreakspace}
\newcommand\BabelDateDot{.\@}  % TODO. \let instead of repeating
\newcommand\BabelDated[1]{{\number#1}}
\newcommand\BabelDatedd[1]{{\ifnum#1<10 0\fi\number#1}}
```

```
\newcommand\BabelDateM[1]{{\number#1}}
\newcommand\BabelDateMM[1]{{\ifnum#1<10 0\fi\number#1}}
\newcommand\BabelDateMMMM[1]{{%
  \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
\newcommand\BabelDatey[1]{{\number#1}}%
\newcommand\BabelDateyy[1]{{%
  \ifnum#1<10 0\number#1 %
  \else\ifnum#1<100 \number#1 %
  \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
  \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
  \else
    \bbl@error{limit-two-digits}{}{}{}%
  \fi\fi\fi\fi}}
\newcommand\BabelDateyyyy[1]{{\number#1}} % TODO - add leading 0
\newcommand\BabelDateU[1]{{\number#1}}%
\def\bbl@replace@finish@iii#1{%
  \bbl@exp{\def\\#1####1####2####3{\the\toks@}}}
\def\bbl@TG@@date{%
  \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
  \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
  \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
  \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
  \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
  \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
  \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
  \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
  \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
  \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
  \bbl@replace\bbl@toreplace{[U]}{\BabelDateU{####1}}%
  \bbl@replace\bbl@toreplace{[y|}{\bbl@datecntr[####1|}%
  \bbl@replace\bbl@toreplace{[U|}{\bbl@datecntr[####1|}%
  \bbl@replace\bbl@toreplace{[m|}{\bbl@datecntr[####2|}%
  \bbl@replace\bbl@toreplace{[d|}{\bbl@datecntr[####3|}%
  \bbl@replace@finish@iii\bbl@toreplace}
\def\bbl@datecntr{\expandafter\bbl@xdatecntr\expandafter}
\def\bbl@xdatecntr[#1|#2]{\localenumeral{#2}{#1}}
```

Transforms.

```
\bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
\bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
\def\bbl@transforms@aux#1#2#3#4,#5\relax{%
  #1[#2]{#3}{#4}{#5}}
\begingroup %  A hack. TODO. Don't require a specific order
  \catcode`\%=12
  \catcode`\&=14
  \gdef\bbl@transforms#1#2#3{&%
    \directlua{
       local str = [==[#2]==]
       str = str:gsub('%.%d+%.%d+$', '')
       token.set_macro('babeltempa', str)
    }&%
    \def\babeltempc{}&%
    \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
    \ifin@\else
      \bbl@xin@{:\babeltempa,}{,\bbl@KVP@transforms,}&%
    \fi
    \ifin@
      \bbl@foreach\bbl@KVP@transforms{&%
        \bbl@xin@{:\babeltempa,}{,##1,}&%
        \ifin@  &% font:font:transform syntax
          \directlua{
            local t = {}
            for m in string.gmatch('##1'..':', '(.-):') do
              table.insert(t, m)
            end
            table.remove(t)
            token.set_macro('babeltempc', ',fonts=' .. table.concat(t, ' '))
          }&%
        \fi}&%
      \in@{.0$}{#2$}&%
      \ifin@
        \directlua{&% (\attribute) syntax
          local str = string.match([[\bbl@KVP@transforms]],
                        '%(([^%(]-)%)[^%)]-\babeltempa')
          if str == nil then
            token.set_macro('babeltempb', '')
          else
            token.set_macro('babeltempb', ',attribute=' .. str)
          end
        }&%
        \toks@{#3}&%
        \bbl@exp{&%
          \\\g@addto@macro\\\bbl@release@transforms{&%
            \relax  &% Closes previous \bbl@transforms@aux
            \\\bbl@transforms@aux
              \\#1{label=\babeltempa\babeltempb\babeltempc}&%
                 {\languagename}{\the\toks@}}}&%
      \else
        \g@addto@macro\bbl@release@transforms{, {#3}}&%
      \fi
    \fi}
\endgroup
```

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

```
\def\bbl@provide@lsys#1{%
  \bbl@ifunset{bbl@lname@#1}%
    {\bbl@load@info{#1}}%
    {}%
  \bbl@csarg\let{lsys@#1}\@empty
  \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
  \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
  \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
  \bbl@ifunset{bbl@lname@#1}{}%
    {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
  \ifcase\bbl@engine\or\or
    \bbl@ifunset{bbl@prehc@#1}{}%
      {\bbl@exp{\\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
        {}%
        {\ifx\bbl@xenohyph\@undefined
           \global\let\bbl@xenohyph\bbl@xenohyph@d
           \ifx\AtBeginDocument\@notprerr
             \expandafter\@secondoftwo  % to execute right now
           \fi
           \AtBeginDocument{%
             \bbl@patchfont{\bbl@xenohyph}%
             {\expandafter\select@language\expandafter{\languagename}}}%
        \fi}}%
  \fi
  \bbl@csarg\bbl@toglobal{lsys@#1}}
\def\bbl@xenohyph@d{%
  \bbl@ifset{bbl@prehc@\languagename}%
    {\ifnum\hyphenchar\font=\defaulthyphenchar
       \iffontchar\font\bbl@cl{prehc}\relax
         \hyphenchar\font\bbl@cl{prehc}\relax
       \else\iffontchar\font"200B
```

```
        \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's\\%
           'HyphenChar' to another value, but be aware\\%
           this setting is not safe (see the manual).\\%
           Reported}%
        \hyphenchar\font\defaulthyphenchar
      \fi\fi
    \fi}%
   {\hyphenchar\font\defaulthyphenchar}}
  % \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}%          boxed, to avoid extra spaces:
  {\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\<bbl@cntr@digits@\languagename>\<\languagename digits>%
    \def\<\languagename counter>####1{%      ie, \langcounter
      \\\expandafter\<bbl@counter@\languagename>%
      \\\csname c@####1\endcsname}%
    \def\<bbl@counter@\languagename>####1{% ie, \bbl@counter@lang
      \\\expandafter\<bbl@digits@\languagename>%
      \\\number####1\\\@nil}}%
  \def\bbl@tempa##1##2##3##4##5{%
    \bbl@exp{%    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##1%
         \\\else\\\ifx6########1##2%
         \\\else\\\ifx7########1##3%
         \\\else\\\ifx8########1##4%
         \\\else\\\ifx9########1##5%
         \\\else########1%
         \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
         \\\expandafter\<bbl@digits@\languagename>%
       \\\fi}}}%
  \bbl@tempa}
```

Alphabetic counters must be converted from a space separated list to an \ifcase structure.

```
\def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}
```

```
  \ifx\\#1%             % \\ before, in case #1 is multiletter
    \bbl@exp{%
      \def\\\bbl@tempa####1{%
        \<ifcase>####1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
  \else
    \toks@\expandafter{\the\toks@\or #1}%
    \expandafter\bbl@buildifcase
  \fi}
```

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
  \expandafter{\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\@@#9{%
  \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@\languagename}%
    {\bbl@cs{cntr@#1.4@\languagename}#5%
     \bbl@cs{cntr@#1.3@\languagename}#6%
     \bbl@cs{cntr@#1.2@\languagename}#7%
     \bbl@cs{cntr@#1.1@\languagename}#8%
     \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
       \bbl@ifunset{bbl@cntr@#1.S.321@\languagename}{}%
         {\bbl@cs{cntr@#1.S.321@\languagename}}%
     \fi}%
    {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\languagename}}}
\def\bbl@alphnum@invalid#1{%
  \bbl@error{alphabetic-too-large}{#1}{}{}}
```

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

```
\def\bbl@localeinfo#1#2{%
  \bbl@ifunset{bbl@info@#2}{#1}%
    {\bbl@ifunset{bbl@\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{}%
  \else
    \bbl@localeinfo
      {\bbl@error{no-ini-info}{}{}{}}%
      {#1}%
  \fi}
% \@namedef{bbl@info@name.locale}{lcname}
\@namedef{bbl@info@tag.ini}{lini}
\@namedef{bbl@info@name.english}{elname}
\@namedef{bbl@info@name.opentype}{lname}
\@namedef{bbl@info@tag.bcp47}{tbcp}
\@namedef{bbl@info@language.tag.bcp47}{lbcp}
\@namedef{bbl@info@tag.opentype}{lotf}
```

```
\@namedef{bbl@info@script.name}{esname}
\@namedef{bbl@info@script.name.opentype}{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 `\BCPdata` to be defined. While `language`, `region`, `script`, and `variant` are recognized, `extension.`⟨*s*⟩ for singletons may change.

```
\ifcase\bbl@engine % Converts utf8 to its code (expandable)
  \def\bbl@utftocode#1{\the\numexpr\decode@UTFviii#1\relax}
\else
  \def\bbl@utftocode#1{\expandafter`\string#1}
\fi
% Still somewhat hackish. WIP. Note |\str_if_eq:nnTF| is fully
% expandable (|\bbl@ifsamestring| isn't).
\providecommand\BCPdata{}
\ifx\renewcommand\@undefined\else % For plain. TODO. It's a quick fix
  \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty}
  \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
    \@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
      {\bbl@bcpdata@ii{#6}\bbl@main@language}%
      {\bbl@bcpdata@ii{#1#2#3#4#5#6}\languagename}}%
  \def\bbl@bcpdata@ii#1#2{%
    \bbl@ifunset{bbl@info@#1.tag.bcp47}%
      {\bbl@error{unknown-ini-field}{#1}{}{}}%
      {\bbl@ifunset{bbl@\csname bbl@info@#1.tag.bcp47\endcsname @#2}{}%
        {\bbl@cs{\csname bbl@info@#1.tag.bcp47\endcsname @#2}}}}
\fi
\@namedef{bbl@info@casing.tag.bcp47}{casing}
\newcommand\BabelUppercaseMapping[3]{%
  \DeclareUppercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
\newcommand\BabelTitlecaseMapping[3]{%
  \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
\newcommand\BabelLowercaseMapping[3]{%
  \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
```

The parser for `casing` and `casing.`⟨*variant*⟩.

```
\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@tempa\bbl@tempc\@empty}
\def\bbl@casemapping@i#1{%
  \def\bbl@tempb{#1}%
  \ifcase\bbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
    \@nameuse{regex_replace_all:nnN}%
      {[\x{c0}-\x{ff}][\x{80}-\x{bf}]*}{{\0}}\bbl@tempb
  \else
    \@nameuse{regex_replace_all:nnN}{.}{{\0}}\bbl@tempb % TODO. needed?
  \fi
  \expandafter\bbl@casemapping@ii\bbl@tempb\@@}
\def\bbl@casemapping@ii#1#2#3\@@{%
  \in@{#1#3}{<>}% ie, if <u>, <l>, <t>
  \ifin@
    \edef\bbl@tempe{%
      \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
  \else
    \ifcase\bbl@tempe\relax
      \DeclareUppercaseMapping[\bbl@templ]{\bbl@utftocode{#1}}{#2}%
      \DeclareLowercaseMapping[\bbl@templ]{\bbl@utftocode{#2}}{#1}%
    \or
      \DeclareUppercaseMapping[\bbl@templ]{\bbl@utftocode{#1}}{#2}%
    \or
      \DeclareLowercaseMapping[\bbl@templ]{\bbl@utftocode{#1}}{#2}%
    \or
      \DeclareTitlecaseMapping[\bbl@templ]{\bbl@utftocode{#1}}{#2}%
    \fi
  \fi}
```

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

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

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

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

5 Adjusting the Babel behavior

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

```
\newcommand\babeladjust[1]{%  TODO. Error handling.
```

```
  \bbl@forkv{#1}{%
    \bbl@ifunset{bbl@ADJ@##1@##2}%
      {\bbl@cs{ADJ@##1}{##2}}%
      {\bbl@cs{ADJ@##1@##2}}}}
%
\def\bbl@adjust@lua#1#2{%
  \ifvmode
    \ifnum\currentgrouplevel=\z@
      \directlua{ Babel.#2 }%
      \expandafter\expandafter\expandafter\@gobble
    \fi
  \fi
  {\bbl@error{adjust-only-vertical}{#1}{}{}}}% Gobbled if everything went ok.
\@namedef{bbl@ADJ@bidi.mirroring@on}{%
  \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
\@namedef{bbl@ADJ@bidi.mirroring@off}{%
  \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
\@namedef{bbl@ADJ@bidi.text@on}{%
  \bbl@adjust@lua{bidi}{bidi_enabled=true}}
\@namedef{bbl@ADJ@bidi.text@off}{%
  \bbl@adjust@lua{bidi}{bidi_enabled=false}}
\@namedef{bbl@ADJ@bidi.math@on}{%
  \let\bbl@noamsmath\@empty}
\@namedef{bbl@ADJ@bidi.math@off}{%
  \let\bbl@noamsmath\relax}
%
\@namedef{bbl@ADJ@bidi.mapdigits@on}{%
  \bbl@adjust@lua{bidi}{digits_mapped=true}}
\@namedef{bbl@ADJ@bidi.mapdigits@off}{%
  \bbl@adjust@lua{bidi}{digits_mapped=false}}
%
\@namedef{bbl@ADJ@linebreak.sea@on}{%
  \bbl@adjust@lua{linebreak}{sea_enabled=true}}
\@namedef{bbl@ADJ@linebreak.sea@off}{%
  \bbl@adjust@lua{linebreak}{sea_enabled=false}}
\@namedef{bbl@ADJ@linebreak.cjk@on}{%
  \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
\@namedef{bbl@ADJ@linebreak.cjk@off}{%
  \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
\@namedef{bbl@ADJ@justify.arabic@on}{%
  \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
\@namedef{bbl@ADJ@justify.arabic@off}{%
  \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
%
\def\bbl@adjust@layout#1{%
  \ifvmode
    #1%
    \expandafter\@gobble
  \fi
  {\bbl@error{layout-only-vertical}{}{}{}}}% Gobbled if everything went ok.
\@namedef{bbl@ADJ@layout.tabular@on}{%
  \ifnum\bbl@tabular@mode=\tw@
    \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}%
  \else
    \chardef\bbl@tabular@mode\@ne
  \fi}
\@namedef{bbl@ADJ@layout.tabular@off}{%
  \ifnum\bbl@tabular@mode=\tw@
    \bbl@adjust@layout{\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
  \let\bbl@savelastskip\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}
```

5.1 Cross referencing macros

The LaTeX book states:

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

When the above quote should still be true when a document is typeset in a language that has active characters, special care has to be taken of the category codes of these characters when they appear in an argument of the cross referencing macros.
When a cross referencing command processes its argument, all tokens in this argument should be character tokens with category 'letter' or 'other'.
The following package options control which macros are to be redefined.

```
⟨⟨*More package options⟩⟩ ≡
\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}}
⟨⟨/More package options⟩⟩
```

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

```
\bbl@trace{Cross referencing macros}
\ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
  \def\@newl@bel#1#2#3{%
   {\@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}}}
```

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

```
  \CheckCommand*\@testdef[3]{%
    \def\reserved@a{#3}%
    \expandafter\ifx\csname#1@#2\endcsname\reserved@a
    \else
      \@tempswatrue
    \fi}
```

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.

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

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

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

\@citex The macro used to cite from a bibliography, `\cite`, uses an internal macro, `\@citex`. It is this internal macro that picks up the argument(s), so we redefine this internal macro and leave `\cite` alone. The first argument is used for typesetting, so the shorthands need only be deactivated in the second argument.

```
\bbl@xin@{B}\bbl@opt@safe
\ifin@
  \bbl@redefine\@citex[#1]#2{%
    \@safe@activestrue\edef\bbl@tempa{#2}\@safe@activesfalse
    \org@@citex[#1]{\bbl@tempa}}
```

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

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

Notice that we use `\def` here instead of `\bbl@redefine` because `\org@@citex` is already defined and we don't want to overwrite that definition (it would result in parameter stack overflow because of a circular definition).
(Recent versions of `natbib` change dynamically `\@citex`, so PR4087 doesn't seem fixable in a simple way. Just load `natbib` before.)

```
    \def\@citex[#1][#2]#3{%
      \@safe@activestrue\edef\bbl@tempa{#3}\@safe@activesfalse
      \org@@citex[#1][#2]{\bbl@tempa}}%
    }{}}
```

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

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

`\nocite` 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}
```

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

```
  \bbl@redefine\bibcite{%
    \bbl@cite@choice
    \bibcite}
```

`\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.

```
  \bbl@redefine\@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
       \expandafter\select@language@x\expandafter{\bbl@main@language}%
       \let\protect\noexpand
       \ifcase\bbl@bidimode\else % Only with bidi. See also above
         \edef\thepage{%
           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
       \fi}%
    \fi}
  {\ifbbl@single\else
     \bbl@ifunset{markright }\bbl@redefine\bbl@redefinerobust
     \markright#1{%
       \bbl@ifblank{#1}%
         {\org@markright{}}%
         {\toks@{#1}%
          \bbl@exp{%
            \\\org@markright{\\\protect\\\foreignlanguage{\languagename}%
              {\\\protect\\\bbl@restore@actives\the\toks@}}}}}%
```

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

```
   \ifx\@mkboth\markboth
     \def\bbl@tempc{\let\@mkboth\markboth}%
   \else
     \def\bbl@tempc{}%
   \fi
   \bbl@ifunset{markboth }\bbl@redefine\bbl@redefinerobust
   \markboth#1#2{%
     \protected@edef\bbl@tempb##1{%
       \protect\foreignlanguage
       {\languagename}{\protect\bbl@restore@actives##1}}%
     \bbl@ifblank{#1}%
       {\toks@{}}%
       {\toks@\expandafter{\bbl@tempb{#1}}}%
     \bbl@ifblank{#2}%
       {\@temptokena{}}%
       {\@temptokena\expandafter{\bbl@tempb{#2}}}%
     \bbl@exp{\\\org@markboth{\the\toks@}{\the\@temptokena}}}%
     \bbl@tempc
   \fi}  % end ifbbl@single, end \IfBabelLayout
```

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}{%
```

```
        \bbl@redefine@long\ifthenelse#1#2#3{%
          \let\bbl@temp@pref\pageref
          \let\pageref\org@pageref
          \let\bbl@temp@ref\ref
          \let\ref\org@ref
          \@safe@activestrue
          \org@ifthenelse{#1}%
            {\let\pageref\bbl@temp@pref
             \let\ref\bbl@temp@ref
             \@safe@activesfalse
             #2}%
            {\let\pageref\bbl@temp@pref
             \let\ref\bbl@temp@ref
             \@safe@activesfalse
             #3}%
          }%
        }{}%
      }
\fi
```

5.3.2 varioref

\@@vpageref \vrefpagenum \Ref 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}
      }{}%
    }
\fi
```

5.3.3 hhline

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

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

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

```
\def\substitutefontfamily#1#2#3{%
  \lowercase{\immediate\openout15=#1#2.fd\relax}%
  \immediate\write15{%
    \string\ProvidesFile{#1#2.fd}%
    [\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}{}^^J
    \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
    \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
    \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
    \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
    \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
    \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
    \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
    }%
  \closeout15
  }
\@onlypreamble\substitutefontfamily
```

5.4 Encoding and fonts

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

\ensureascii

```
\bbl@trace{Encoding and fonts}
\newcommand\BabelNonASCII{LGR,LGI,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
\newcommand\BabelNonText{TS1,T3,TS3}
\let\org@TeX\TeX
\let\org@LaTeX\LaTeX
\let\ensureascii\@firstofone
\let\asciiencoding\@empty
\AtBeginDocument{%
  \def\@elt#1{,#1,}%
  \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
  \let\@elt\relax
  \let\bbl@tempb\@empty
  \def\bbl@tempc{OT1}%
  \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
    \bbl@ifunset{T@#1}{}{\def\bbl@tempb{#1}}}%
  \bbl@foreach\bbl@tempa{%
    \bbl@xin@{,#1,}{,\BabelNonASCII,}%
    \ifin@
      \def\bbl@tempb{#1}% Store last non-ascii
    \else\bbl@xin@{,#1,}{,\BabelNonText,}% Pass
      \ifin@\else
        \def\bbl@tempc{#1}% Store last ascii
      \fi
    \fi}%
  \ifx\bbl@tempb\@empty\else
    \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
    \ifin@\else
      \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
    \fi
    \let\asciiencoding\bbl@tempc
    \renewcommand\ensureascii[1]{%
```

```
      {\fontencoding{\asciiencoding}\selectfont#1}}%
    \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
    \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
  \fi}
```

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

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

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

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

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

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

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

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

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

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

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

5.5 Basic bidi support

Work in progress. This code is currently placed here for practical reasons. It will be moved to the correct place soon, I hope.
It is loosely based on `rlbabel.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 `rlbabel` 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 LuaTEX-ja shows, vertical typesetting is possible, too.

```
\bbl@trace{Loading basic (internal) bidi support}
\ifodd\bbl@engine
\else % TODO. Move to txtbabel
  \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200 % Any xe+lua bidi=
    \bbl@error{bidi-only-lua}{}{}{}%
    \let\bbl@beforeforeign\leavevmode
    \AtEndOfPackage{%
      \EnableBabelHook{babel-bidi}%
      \bbl@xebidipar}
  \fi\fi
  \def\bbl@loadxebidi#1{%
    \ifx\RTLfootnotetext\@undefined
      \AtEndOfPackage{%
        \EnableBabelHook{babel-bidi}%
        \bbl@loadfontspec % bidi needs fontspec
        \usepackage#1{bidi}%
        \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
        \def\DigitsDotDashInterCharToks{% See the 'bidi' package
          \ifnum\@nameuse{bbl@wdir@\languagename}=\tw@ % 'AL' bidi
            \bbl@digitsdotdash % So ignore in 'R' bidi
          \fi}}%
    \fi}
  \ifnum\bbl@bidimode>200 % Any xe bidi=
    \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
      \bbl@tentative{bidi=bidi}
      \bbl@loadxebidi{}
    \or
      \bbl@loadxebidi{[rldocument]}
    \or
      \bbl@loadxebidi{}
    \fi
  \fi
\fi
% TODO? Separate:
\ifnum\bbl@bidimode=\@ne % bidi=default
  \let\bbl@beforeforeign\leavevmode
  \ifodd\bbl@engine % lua
    \newattribute\bbl@attr@dir
    \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
    \bbl@exp{\output{\bodydir\pagedir\the\output}}
  \fi
  \AtEndOfPackage{%
    \EnableBabelHook{babel-bidi}% pdf/lua/xe
    \ifodd\bbl@engine\else % pdf/xe
      \bbl@xebidipar
    \fi}
\fi
```

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{% TODO. Base on codes ??
  ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
  Old Hungarian,Lydian,Mandaean,Manichaean,%
  Meroitic Cursive,Meroitic,Old North Arabian,%
  Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
  Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
  Old South Arabian,}%
\def\bbl@provide@dirs#1{%
  \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@
    \fi
  \else
    \global\bbl@csarg\chardef{wdir@#1}\z@
  \fi
  \ifodd\bbl@engine
    \bbl@csarg\ifcase{wdir@#1}%
      \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
    \or
      \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
    \or
      \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
    \fi
  \fi}
\def\bbl@switchdir{%
  \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
  \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
  \bbl@exp{\\\bbl@setdirs\bbl@cl{wdir}}}
\def\bbl@setdirs#1{% TODO - math
  \ifcase\bbl@select@type % TODO - strictly, not the right test
    \bbl@bodydir{#1}%
    \bbl@pardir{#1}% <- Must precede \bbl@textdir
  \fi
  \bbl@textdir{#1}}
% TODO. Only if \bbl@bidimode > 0?:
\AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
\DisableBabelHook{babel-bidi}
```

Now the engine-dependent macros. TODO. Must be moved to the engine files.

```
\ifodd\bbl@engine  % luatex=1
\else % pdftex=0, xetex=2
  \newcount\bbl@dirlevel
  \chardef\bbl@thetextdir\z@
  \chardef\bbl@thepardir\z@
  \def\bbl@textdir#1{%
    \ifcase#1\relax
       \chardef\bbl@thetextdir\z@
       \@nameuse{setlatin}%
       \bbl@textdir@i\beginL\endL
     \else
       \chardef\bbl@thetextdir\@ne
       \@nameuse{setnonlatin}%
       \bbl@textdir@i\beginR\endR
    \fi}
  \def\bbl@textdir@i#1#2{%
    \ifhmode
```

```
      \ifnum\currentgrouplevel>\z@
        \ifnum\currentgrouplevel=\bbl@dirlevel
          \bbl@error{multiple-bidi}{}{}{}%
          \bgroup\aftergroup#2\aftergroup\egroup
        \else
          \ifcase\currentgrouptype\or % 0 bottom
            \aftergroup#2% 1 simple {}
          \or
            \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
          \or
            \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
          \or\or\or % vbox vtop align
          \or
            \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
          \or\or\or\or\or\or % output math disc insert vcent mathchoice
          \or
            \aftergroup#2% 14 \begingroup
          \else
            \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
          \fi
        \fi
        \bbl@dirlevel\currentgrouplevel
      \fi
      #1%
    \fi}
  \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
  \let\bbl@bodydir\@gobble
  \let\bbl@pagedir\@gobble
  \def\bbl@dirparastext{\chardef\bbl@thepardir\bbl@thetextdir}
```

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@xebidipar{%
    \let\bbl@xebidipar\relax
    \TeXXeTstate\@ne
    \def\bbl@xeeverypar{%
      \ifcase\bbl@thepardir
        \ifcase\bbl@thetextdir\else\beginR\fi
      \else
        {\setbox\z@\lastbox\beginR\box\z@}%
      \fi}%
    \let\bbl@severypar\everypar
    \newtoks\everypar
    \everypar=\bbl@severypar
    \bbl@severypar{\bbl@xeeverypar\the\everypar}}
  \ifnum\bbl@bidimode>200 % Any xe bidi=
    \let\bbl@textdir@i\@gobbletwo
    \let\bbl@xebidipar\@empty
    \AddBabelHook{bidi}{foreign}{%
      \ifcase\bbl@thetextdir
        \BabelWrapText{\LR{##1}}%
      \else
        \BabelWrapText{\RL{##1}}%
      \fi}
    \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
  \fi
\fi
```

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

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

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.

```
\bbl@trace{Local Language Configuration}
\ifx\loadlocalcfg\@undefined
  \@ifpackagewith{babel}{noconfigs}%
    {\let\loadlocalcfg\@gobble}%
    {\def\loadlocalcfg#1{%
      \InputIfFileExists{#1.cfg}%
        {\typeout{*************************************^^J%
                       * Local config file #1.cfg used^^J%
                       *}}%
        \@empty}}
\fi
```

5.7 Language options

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

```
\bbl@trace{Language options}
\let\bbl@afterlang\relax
\let\BabelModifiers\relax
\let\bbl@loaded\@empty
\def\bbl@load@language#1{%
  \InputIfFileExists{#1.ldf}%
    {\edef\bbl@loaded{\CurrentOption
       \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
     \expandafter\let\expandafter\bbl@afterlang
        \csname\CurrentOption.ldf-h@@k\endcsname
     \expandafter\let\expandafter\BabelModifiers
        \csname bbl@mod@\CurrentOption\endcsname
     \bbl@exp{\\\AtBeginDocument{%
       \\\bbl@usehooks@lang{\CurrentOption}{begindocument}{{\CurrentOption}}}}}%
    {\IfFileExists{babel-#1.tex}%
      {\def\bbl@tempa{%
         .\\There is a locale ini file for this language.\\%
         If it’s the main language, try adding `provide=*'\\%
         to the babel package options}}%
      {\let\bbl@tempa\empty}%
     \bbl@error{unknown-package-option}{}{}{}}}
```

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

```
\def\bbl@try@load@lang#1#2#3{%
  \IfFileExists{\CurrentOption.ldf}%
    {\bbl@load@language{\CurrentOption}}%
    {#1\bbl@load@language{#2}#3}}
%
\DeclareOption{hebrew}{%
  \ifcase\bbl@engine\or
    \bbl@error{only-pdftex-lang}{hebrew}{luatex}{}%
  \fi
  \input{rlbabel.def}%
  \bbl@load@language{hebrew}}
\DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
\DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
\DeclareOption{polutonikogreek}{%
  \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
\DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
\DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
\DeclareOption{uppersorbian}{\bbl@try@load@lang{}{usorbian}{}}
```

Another way to extend the list of ‘known’ options for babel was to create the file `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.

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

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

```
\ifx\bbl@opt@main\@nnil
  \ifnum\bbl@iniflag>\z@  % if all ldf's: set implicitly, no main pass
    \let\bbl@tempb\@empty
    \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
    \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
    \bbl@foreach\bbl@tempb{%    \bbl@tempb is a reversed list
      \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
        \ifodd\bbl@iniflag % = *=
          \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
        \else % n +=
          \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
        \fi
      \fi}%
  \fi
\else
  \bbl@info{Main language set with 'main='. Except if you have\\%
            problems, prefer the default mechanism for setting\\%
            the main language, ie, as the last declared.\\%
            Reported}
\fi
```

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

```
\ifx\bbl@opt@main\@nnil\else
  \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
  \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
\fi
```

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

```
\bbl@foreach\bbl@language@opts{%
  \def\bbl@tempa{#1}%
  \ifx\bbl@tempa\bbl@opt@main\else
    \ifnum\bbl@iniflag<\tw@    % 0 ø (other = ldf)
      \bbl@ifunset{ds@#1}%
        {\DeclareOption{#1}{\bbl@load@language{#1}}}%
        {}%
    \else                      % + * (other = ini)
      \DeclareOption{#1}{%
        \bbl@ldfinit
        \babelprovide[import]{#1}%
        \bbl@afterldf{}}%
    \fi
  \fi}
\bbl@foreach\@classoptionslist{%
  \def\bbl@tempa{#1}%
  \ifx\bbl@tempa\bbl@opt@main\else
    \ifnum\bbl@iniflag<\tw@    % 0 ø (other = ldf)
      \bbl@ifunset{ds@#1}%
        {\IfFileExists{#1.ldf}%
          {\DeclareOption{#1}{\bbl@load@language{#1}}}%
          {}}%
        {}%
     \else                      % + * (other = ini)
       \IfFileExists{babel-#1.tex}%
         {\DeclareOption{#1}{%
            \bbl@ldfinit
            \babelprovide[import]{#1}%
            \bbl@afterldf{}}}%
         {}%
     \fi
  \fi}
```

And we are done, because all options for this pass has been declared. Those already processed in the first pass are just ignored.
The options have to be processed in the order in which the user specified them (but remember class options are processes before):

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

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

```
\bbl@trace{Option 'main'}
\ifx\bbl@opt@main\@nnil
  \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
  \let\bbl@tempc\@empty
  \edef\bbl@templ{,\bbl@loaded,}
  \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
  \bbl@for\bbl@tempb\bbl@tempa{%
    \edef\bbl@tempd{,\bbl@tempb,}%
    \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
    \bbl@xin@{\bbl@tempd}{\bbl@templ}%
    \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
  \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
  \expandafter\bbl@tempa\bbl@loaded,\@nnil
  \ifx\bbl@tempb\bbl@tempc\else
    \bbl@warning{%
      Last declared language option is '\bbl@tempc',\\%
      but the last processed one was '\bbl@tempb'.\\%
      The main language can't be set as both a global\\%
      and a package option. Use 'main=\bbl@tempc' as\\%
      option. Reported}
  \fi
\else
  \ifodd\bbl@iniflag  % case 1,3 (main is ini)
    \bbl@ldfinit
    \let\CurrentOption\bbl@opt@main
    \bbl@exp{%  \bbl@opt@provide = empty if *
       \\\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
    \bbl@afterldf{}
    \DeclareOption{\bbl@opt@main}{}
  \else % case 0,2 (main is ldf)
    \ifx\bbl@loadmain\relax
      \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
    \else
      \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
    \fi
    \ExecuteOptions{\bbl@opt@main}
    \@namedef{ds@\bbl@opt@main}{}%
  \fi
  \DeclareOption*{}
  \ProcessOptions*
\fi
\bbl@exp{%
  \\\AtBeginDocument{\\\bbl@usehooks@lang{/}{begindocument}{{}}}}%
\def\AfterBabelLanguage{\bbl@error{late-after-babel}{}{}{}}
```

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

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

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

The kernel of the babel system is currently stored in `babel.def`. The file `babel.def` contains most of the code. The file `hyphen.cfg` is a file that can be loaded into the format, which is necessary when you want to be able to switch hyphenation patterns.
Because plain TEX users might want to use some of the features of the babel system too, care has to be taken that plain TEX can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain TEX and LATEX, some of it is for the LATEX case only.
Plain formats based on etex (etex, xetex, luatex) don't load `hyphen.cfg` but `etex.src`, which follows a different naming convention, so we need to define the babel names. It presumes `language.def` exists and it is the same file used when formats were created.
A proxy file for switch.def

```
⟨*kernel⟩
\let\bbl@onlyswitch\@empty
\input babel.def
\let\bbl@onlyswitch\@undefined
⟨/kernel⟩
%
% \section{Error messages}
```

```
%
% They are loaded when |\bll@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.
%
⟨*errors⟩
\catcode`\{=1  \catcode`\}=2  \catcode`\#=6
\catcode`\:=12 \catcode`\,=12 \catcode`\.=12 \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\\{^^J(babel) }%
      \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}}}
% Implicit #2#3#4:
\gdef\bbl@error#1{\csname bbl@err@#1\endcsname}
%
\bbl@errmessage{not-yet-available}
    {Not yet available}%
    {Find an armchair, sit down and wait}
\bbl@errmessage{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@errmessage{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@errmessage{undefined-language}
   {You haven't defined the language '#1' yet.\\%
    Perhaps you misspelled it or your installation\\%
    is not complete}%
   {Your command will be ignored, type <return> to proceed}
\bbl@errmessage{shorthand-is-off}
   {I 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@errmessage{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}%
   {You may proceed, but expect unexpected results}
```

```
\bbl@errmessage{not-a-shorthand-b}
   {I can't switch '\string#2' on or off--not a shorthand}%
   {This character is not a shorthand. Maybe you made\\%
    a typing mistake? I will ignore your instruction.}
\bbl@errmessage{unknown-attribute}
   {The attribute #2 is unknown for language #1.}%
   {Your command will be ignored, type <return> to proceed}
\bbl@errmessage{missing-group}
   {Missing group for string \string#1}%
   {You must assign strings to some category, typically\\%
    captions or extras, but you set none}
\bbl@errmessage{only-lua-xe}
   {This macro is available only in LuaLaTeX and XeLaTeX.}%
   {Consider switching to these engines.}
\bbl@errmessage{only-lua}
   {This macro is available only in LuaLaTeX.}%
   {Consider switching to that engine.}
\bbl@errmessage{unknown-provide-key}
   {Unknown key '#1' in \string\babelprovide}%
   {See the manual for valid keys}%
\bbl@errmessage{unknown-mapfont}
   {Option '\bbl@KVP@mapfont' unknown for\\%
    mapfont. Use 'direction'.}%
   {See the manual for details.}
\bbl@errmessage{no-ini-file}
   {There is no ini file for the requested language\\%
    (#1: \languagename). Perhaps you misspelled it or your\\%
    installation is not complete.}%
   {Fix the name or reinstall babel.}
\bbl@errmessage{digits-is-reserved}
   {The counter name 'digits' is reserved for mapping\\%
    decimal digits}%
   {Use another name.}
\bbl@errmessage{limit-two-digits}
   {Currently two-digit years are restricted to the\\
    range 0-9999.}%
   {There is little you can do. Sorry.}
\bbl@errmessage{alphabetic-too-large}
 {Alphabetic numeral too large (#1)}%
 {Currently this is the limit.}
\bbl@errmessage{no-ini-info}
   {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.}
\bbl@errmessage{unknown-ini-field}
   {Unknown field '#1' in \string\BCPdata.\\%
    Perhaps you misspelled it.}%
   {See the manual for details.}
\bbl@errmessage{unknown-locale-key}
   {Unknown key for locale '#2':\\%
    #3\\%
    \string#1 will be set to \relax}%
   {Perhaps you misspelled it.}%
\bbl@errmessage{adjust-only-vertical}
   {Currently, #1 related features can be adjusted only\\%
    in the main vertical list.}%
   {Maybe things change in the future, but this is what it is.}
\bbl@errmessage{layout-only-vertical}
   {Currently, layout related features can be adjusted only\\%
    in vertical mode.}%
   {Maybe things change in the future, but this is what it is.}
\bbl@errmessage{bidi-only-lua}
```

```
  {The bidi method 'basic' is available only in\\%
   luatex. I'll continue with 'bidi=default', so\\%
   expect wrong results}%
  {See the manual for further details.}
\bbl@errmessage{multiple-bidi}
  {Multiple bidi settings inside a group}%
  {I'll insert a new group, but expect wrong results.}
\bbl@errmessage{unknown-package-option}
  {Unknown option '\CurrentOption'. Either you misspelled it\\%
   or the language definition file \CurrentOption.ldf\\%
   was not found%
   \bbl@tempa}
  {Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
   activeacute, activegrave, noconfigs, safe=, main=, math=\\%
   headfoot=, strings=, config=, hyphenmap=, or a language name.}
\bbl@errmessage{config-not-found}
  {Local config file '\bbl@opt@config.cfg' not found}%
  {Perhaps you misspelled it.}
\bbl@errmessage{late-after-babel}
  {Too late for \string\AfterBabelLanguage}%
  {Languages have been loaded, so I can do nothing}
\bbl@errmessage{double-hyphens-class}
  {Double hyphens aren't allowed in \string\babelcharclass\\%
   because it's potentially ambiguous}%
  {See the manual for further info}
\bbl@errmessage{unknown-interchar}
  {'#1' for '\languagename' cannot be enabled.\\%
   Maybe there is a typo.}%
  {See the manual for further details.}
\bbl@errmessage{unknown-interchar-b}
  {'#1' for '\languagename' cannot be disabled.\\%
   Maybe there is a typo.}%
  {See the manual for further details.}
\bbl@errmessage{charproperty-only-vertical}
  {\string\babelcharproperty\space can be used only in\\%
   vertical mode (preamble or between paragraphs)}%
  {See the manual for further info}
\bbl@errmessage{unknown-char-property}
  {No property named '#2'. Allowed values are\\%
   direction (bc), mirror (bmg), and linebreak (lb)}%
  {See the manual for further info}
\bbl@errmessage{bad-transform-option}
  {Bad option '#1' in a transform.\\%
   I’ll ignore it but expect more errors}%
  {See the manual for further info.}
\bbl@errmessage{font-conflict-transforms}
  {Transforms cannot be re-assigned to different\\%
   fonts. The conflict is in '\bbl@kv@label'.\\%
   Apply the same fonts or use a different label}%
  {See the manual for further details.}
\bbl@errmessage{transform-not-available}
  {'#1' for '\languagename' cannot be enabled.\\%
   Maybe there is a typo or it’s a font-dependent transform}%
  {See the manual for further details.}
\bbl@errmessage{transform-not-available-b}
  {'#1' for '\languagename' cannot be disabled.\\%
   Maybe there is a typo or it’s a font-dependent transform}%
  {See the manual for further details.}
\bbl@errmessage{year-out-range}
  {Year out of range.\\%
   The allowed range is #1}%
  {See the manual for further details.}
\bbl@errmessage{only-pdftex-lang}
```

```
  {The '#1' ldf style doesn't work with #2,\\%
   but you can use the ini locale instead.\\%
   Try adding 'provide=*' to the option list. You may\\%
   also want to set 'bidi=' to some value.}%
  {See the manual for further details.}
⟨/errors⟩
⟨*patterns⟩
```

7 Loading hyphenation patterns

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

```
⟨⟨Make sure ProvidesFile is defined⟩⟩
\ProvidesFile{hyphen.cfg}[⟨⟨date⟩⟩ v⟨⟨version⟩⟩ Babel hyphens]
\xdef\bbl@format{\jobname}
\def\bbl@version{⟨⟨version⟩⟩}
\def\bbl@date{⟨⟨date⟩⟩}
\ifx\AtBeginDocument\@undefined
  \def\@empty{}
\fi
⟨⟨Define core switching macros⟩⟩
```

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

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

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

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

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

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

`\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 `\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 `\`⟨*lang*⟩`hyphenmins` macro. When no assignments were made we provide a default setting.
Some pattern files contain changes to the `\lccode` en `\uccode` arrays. Such changes should remain local to the language; therefore we process the pattern file in a group; the `\patterns` command acts globally so its effect will be remembered.
Then we globally store the settings of `\lefthyphenmin` and `\righthyphenmin` and close the group.
When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)
`\bbl@languages` saves a snapshot of the loaded languages in the form
`\bbl@elt{`⟨*language-name*⟩`}{`⟨*number*⟩`} {`⟨*patterns-file*⟩`}{`⟨*exceptions-file*⟩`}`. Note the last 2 arguments are empty in 'dialects' defined in `language.dat` with =. Note also the language name can have encoding info.
Finally, if the counter `\language` is equal to zero we execute the synonyms stored.

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

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

```
\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
  \else
    \input babel-\bbl@format.cfg\relax
  \fi
  \closein1
\endgroup
\bbl@hook@loadkernel{switch.def}
```

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

```
\openin1 = language.dat
```

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

```
\def\languagename{english}%
\ifeof1
  \message{I couldn't find the file language.dat,\space
           I will try the file hyphen.tex}
  \input hyphen.tex\relax
  \chardef\l@english\z@
\else
```

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

```
  \last@language\m@ne
```

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

```
  \loop
    \endlinechar\m@ne
    \read1 to \bbl@line
    \endlinechar`\^^M
```

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

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

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

```
  \begingroup
    \def\bbl@elt#1#2#3#4{%
      \global\language=#2\relax
      \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{language.dat loads no language, only synonyms}
  \errmessage{Orphan language synonym}
\fi
```

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

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

```
\let\process@synonym\@undefined
\let\process@language\@undefined
\let\bbl@get@enc\@undefined
\let\bbl@hyph@enc\@undefined
\let\bbl@tempa\@undefined
\let\bbl@hook@loadkernel\@undefined
\let\bbl@hook@everylanguage\@undefined
\let\bbl@hook@loadpatterns\@undefined
\let\bbl@hook@loadexceptions\@undefined
⟨/patterns⟩
```

Here the code for iniTEX 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].

```
⟨⟨*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 }
⟨⟨/More package options⟩⟩
```

With explicit languages, we could define the font at once, but we don't. Just wait and see if the language is actually activated. `bbl@font` replaces hardcoded font names inside `\..family` by the corresponding macro `\..default`.
At the time of this writing, fontspec shows a warning about there are languages not available, which some people think refers to babel, even if there is nothing wrong. Here is hack to patch fontspec to avoid the misleading (and mostly unuseful) message.

```
⟨⟨*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][]{%  1=langs/scripts 2=fam
  \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][]{% 1=features 2=fontname, @font=rm|sf|tt
  \bbl@ifunset{\bbl@tempb family}%
```

```
    {\bbl@providefam{\bbl@tempb}}%
    {}%
  % For the default font, just in case:
  \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
  \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
    {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
     \bbl@exp{%
       \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
       \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
                      \<\bbl@tempb default>\<\bbl@tempb family>}}%
    {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
       \bbl@csarg\def{\bbl@tempb dflt@##1}{<>{#1}{#2}}}}}%
```

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

```
\def\bbl@providefam#1{%
  \bbl@exp{%
    \\\newcommand\<#1default>{}% Just define it
    \\\bbl@add@list\\\bbl@font@fams{#1}%
    \\\DeclareRobustCommand\<#1family>{%
      \\\not@math@alphabet\<#1family>\relax
      % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
      \\\fontfamily\<#1default>%
      \<ifx>\\\UseHooks\\\@undefined\<else>\\\UseHook{#1family}\<fi>%
      \\\selectfont}%
    \\\DeclareTextFontCommand{\<text#1>}{\<#1family>}}}
```

The following macro is activated when the hook babel-fontspec is enabled. But before, we define a macro for a warning, which sets a flag to avoid duplicate them.

```
\def\bbl@nostdfont#1{%
  \bbl@ifunset{bbl@WFF@\f@family}%
    {\bbl@csarg\gdef{WFF@\f@family}{}%  Flag, to avoid dupl warns
     \bbl@infowarn{The current font is not a babel standard family:\\%
       #1%
       \fontname\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}}{}%
  \bbl@exp{%  eg Arabic -> arabic
    \lowercase{\edef\\\bbl@tempa{\bbl@cl{sname}}}}%
  \bbl@foreach\bbl@font@fams{%
    \bbl@ifunset{bbl@##1dflt@\languagename}%    (1) language?
      {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}%   (2) from script?
         {\bbl@ifunset{bbl@##1dflt@}%           2=F - (3) from generic?
           {}%                                  123=F - nothing!
           {\bbl@exp{%                          3=T - from generic
              \global\let\<bbl@##1dflt@\languagename>%
                         \<bbl@##1dflt@>}}}%
         {\bbl@exp{%                            2=T - from script
            \global\let\<bbl@##1dflt@\languagename>%
                       \<bbl@##1dflt@*\bbl@tempa>}}}%
      {}}%                               1=T - language, already defined
  \def\bbl@tempa{\bbl@nostdfont{}}%  TODO. Don't use \bbl@tempa
  \bbl@foreach\bbl@font@fams{%     don't gather with prev for
    \bbl@ifunset{bbl@##1dflt@\languagename}%
      {\bbl@cs{famrst@##1}%
       \global\bbl@csarg\let{famrst@##1}\relax}%
      {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
```

```
        \\\bbl@add\\\originalTeX{%
          \\\bbl@font@rst{\bbl@cl{##1dflt}}%
                         \<##1default>\<##1family>{##1}}%
        \\\bbl@font@set\<bbl@##1dflt@\languagename>% the main part!
                       \<##1default>\<##1family>}}}%
  \bbl@ifrestoring{}{\bbl@tempa}}%
```

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

```
\ifx\f@family\@undefined\else   % if latex
  \ifcase\bbl@engine            % if pdftex
    \let\bbl@ckeckstdfonts\relax
  \else
    \def\bbl@ckeckstdfonts{%
      \begingroup
        \global\let\bbl@ckeckstdfonts\relax
        \let\bbl@tempa\@empty
        \bbl@foreach\bbl@font@fams{%
          \bbl@ifunset{bbl@##1dflt@}%
            {\@nameuse{##1family}%
             \bbl@csarg\gdef{WFF@\f@family}{}% Flag
             \bbl@exp{\\\bbl@add\\\bbl@tempa{* \<##1family>= \f@family\\\\%
                \space\space\fontname\font\\\\}}%
             \bbl@csarg\xdef{##1dflt@}{\f@family}%
             \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
            {}}%
        \ifx\bbl@tempa\@empty\else
          \bbl@infowarn{The following font families will use the default\\%
            settings for all or some languages:\\%
            \bbl@tempa
            There is nothing intrinsically wrong with it, but\\%
            'babel' will no set Script and Language, which could\\%
             be relevant in some languages. If your document uses\\%
             these families, consider redefining them with \string\babelfont.\\%
            Reported}%
        \fi
      \endgroup}
  \fi
\fi
```

Now the macros defining the font with fontspec.
When there are repeated keys in fontspec, the last value wins. So, we just place the ini settings at the beginning, and user settings will take precedence. We must deactivate temporarily \bbl@mapselect because \selectfont is called internally when a font is defined.
For historical reasons, LaTeX can select two different series (bx and b), for what is conceptually a single one. This can lead to problems when a single family requires several fonts, depending on the language, mainly because 'substitutions' with some combinations are not done consistently – sometimes bx/sc is the correct font, but sometimes points to b/n, even if b/sc exists. So, some substitutions are redefined (in a somewhat hackish way, by inspecting if the variant declaration contains >ssub*).

```
\def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
  \bbl@xin@{<>}{#1}%
  \ifin@
    \bbl@exp{\\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
  \fi
  \bbl@exp{%              'Unprotected' macros return prev values
    \def\\#2{#1}%         eg, \rmdefault{\bbl@rmdflt@lang}
    \\\bbl@ifsamestring{#2}{\f@family}%
      {\\#3%
       \\\bbl@ifsamestring{\f@series}{\bfdefault}{\\\bfseries}{}%
       \let\\\bbl@tempa\relax}%
      {}}}
%     TODO - next should be global?, but even local does its job. I'm
%     still not sure -- must investigate:
```

```
\def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
  \let\bbl@tempe\bbl@mapselect
  \edef\bbl@tempb{\bbl@stripslash#4/}% Catcodes hack (better pass it).
  \bbl@exp{\\\bbl@replace\\\bbl@tempb{\bbl@stripslash\family/}{}}%
  \let\bbl@mapselect\relax
  \let\bbl@temp@fam#4%       eg, '\rmfamily', to be restored below
  \let#4\@empty      %       Make sure \renewfontfamily is valid
  \bbl@exp{%
    \let\\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
    \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
      {\\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
    \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
      {\\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
    \let\\\bbl@tempfs@nx\<__fontspec_warning:nx>%
    \let\<__fontspec_warning:nx>\\\bbl@fs@warn@nx
    \let\\\bbl@tempfs@nxx\<__fontspec_warning:nxx>%
    \let\<__fontspec_warning:nxx>\\\bbl@fs@warn@nxx
    \\\renewfontfamily\\#4%
      [\bbl@cl{lsys},% xetex removes unknown features :-(
       \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
       #2]}{#3}% ie \bbl@exp{..}{#3}
  \bbl@exp{%
    \let\<__fontspec_warning:nx>\\\bbl@tempfs@nx
    \let\<__fontspec_warning:nxx>\\\bbl@tempfs@nxx}%
  \begingroup
     #4%
     \xdef#1{\f@family}%     eg, \bbl@rmdflt@lang{FreeSerif(0)}
  \endgroup % TODO. Find better tests:
  \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
    {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
  \ifin@
    \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
  \fi
  \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
    {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
  \ifin@
    \global\bbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
  \fi
  \let#4\bbl@temp@fam
  \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
  \let\bbl@mapselect\bbl@tempe}%
```

font@rst and famrst are only used when there is no global settings, to save and restore de previous families. Not really necessary, but done for optimization.

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

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

```
\def\bbl@font@fams{rm,sf,tt}
⟨⟨/Font selection⟩⟩
```

9 Hooks for XeTeX and LuaTeX

9.1 XeTeX

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

```
⟨⟨*Footnote changes⟩⟩ ≡
\bbl@trace{Bidi footnotes}
\ifnum\bbl@bidimode>\z@ % Any bidi=
  \def\bbl@footnote#1#2#3{%
    \@ifnextchar[%
      {\bbl@footnote@o{#1}{#2}{#3}}%
      {\bbl@footnote@x{#1}{#2}{#3}}}
  \long\def\bbl@footnote@x#1#2#3#4{%
    \bgroup
      \select@language@x{\bbl@main@language}%
      \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
    \egroup}
  \long\def\bbl@footnote@o#1#2#3[#4]#5{%
    \bgroup
      \select@language@x{\bbl@main@language}%
      \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
    \egroup}
  \def\bbl@footnotetext#1#2#3{%
    \@ifnextchar[%
      {\bbl@footnotetext@o{#1}{#2}{#3}}%
      {\bbl@footnotetext@x{#1}{#2}{#3}}}
  \long\def\bbl@footnotetext@x#1#2#3#4{%
    \bgroup
      \select@language@x{\bbl@main@language}%
      \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
    \egroup}
  \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
    \bgroup
      \select@language@x{\bbl@main@language}%
      \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
    \egroup}
  \def\BabelFootnote#1#2#3#4{%
    \ifx\bbl@fn@footnote\@undefined
      \let\bbl@fn@footnote\footnote
    \fi
    \ifx\bbl@fn@footnotetext\@undefined
      \let\bbl@fn@footnotetext\footnotetext
    \fi
    \bbl@ifblank{#2}%
      {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
       \@namedef{\bbl@stripslash#1text}%
         {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
      {\def#1{\bbl@exp{\\\bbl@footnote{\\\foreignlanguage{#2}}}{#3}{#4}}%
       \@namedef{\bbl@stripslash#1text}%
         {\bbl@exp{\\\bbl@footnotetext{\\\foreignlanguage{#2}}}{#3}{#4}}}}
\fi
⟨⟨/Footnote changes⟩⟩
```

Now, the code.

```
⟨*xetex⟩
\def\BabelStringsDefault{unicode}
\let\xebbl@stop\relax
\AddBabelHook{xetex}{encodedcommands}{%
  \def\bbl@tempa{#1}%
  \ifx\bbl@tempa\@empty
    \XeTeXinputencoding"bytes"%
  \else
    \XeTeXinputencoding"#1"%
  \fi
  \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
\AddBabelHook{xetex}{stopcommands}{%
  \xebbl@stop
  \let\xebbl@stop\relax}
\def\bbl@input@classes{% Used in CJK intraspaces
  \input{load-unicode-xetex-classes.tex}%
  \let\bbl@input@classes\relax}
\def\bbl@intraspace#1 #2 #3\@@{%
  \bbl@csarg\gdef{xeisp@\languagename}%
```

```
    {\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}{}%
      {\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 % We may override the ini
        \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
      \fi
      \ifx\bbl@KVP@intrapenalty\@nnil\else
        \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
      \fi
      \bbl@exp{%
        % TODO. Execute only once (but redundant):
        \\\bbl@add\<extras\languagename>{%
          \XeTeXlinebreaklocale "\bbl@cl{tbcp}"%
          \<bbl@xeisp@\languagename>%
          \<bbl@xeipn@\languagename>}%
        \\\bbl@toglobal\<extras\languagename>%
        \\\bbl@add\<noextras\languagename>{%
          \XeTeXlinebreaklocale ""}%
        \\\bbl@toglobal\<noextras\languagename>}%
      \ifx\bbl@ispacesize\@undefined
        \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
        \ifx\AtBeginDocument\@notprerr
          \expandafter\@secondoftwo  % to execute right now
        \fi
        \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
      \fi}%
  \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{}
```

10 Support for interchar

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

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

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

```
\AddBabelHook{babel-interchar}{beforeextras}{%
  \@nameuse{bbl@xechars@\languagename}}
\DisableBabelHook{babel-interchar}
\protected\def\bbl@charclass#1{%
  \ifnum\count@<\z@
    \count@-\count@
    \loop
      \bbl@exp{%
        \\\babel@savevariable{\XeTeXcharclass`\Uchar\count@}}%
      \XeTeXcharclass\count@ \bbl@tempc
      \ifnum\count@<`#1\relax
      \advance\count@\@ne
    \repeat
  \else
    \babel@savevariable{\XeTeXcharclass`#1}%
    \XeTeXcharclass`#1 \bbl@tempc
  \fi
  \count@`#1\relax}
```

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

```
\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\IfBabelIntercharT[2]{%
  \bbl@carg\bbl@add{bbl@icsave@\CurrentOption}{\bbl@ifinterchar{#1}{#2}}}%
\newcommand\babelcharclass[3]{%
  \EnableBabelHook{babel-interchar}%
  \bbl@csarg\newXeTeXintercharclass{xeclass@#2@#1}%
  \def\bbl@tempb##1{%
    \ifx##1\@empty\else
      \ifx##1-%
        \bbl@upto
      \else
        \bbl@charclass{%
          \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
      \fi
      \expandafter\bbl@tempb
    \fi}%
  \bbl@ifunset{bbl@xechars@#1}%
    {\toks@{%
       \babel@savevariable\XeTeXinterchartokenstate
       \XeTeXinterchartokenstate\@ne
      }}%
    {\toks@\expandafter\expandafter\expandafter{%
       \csname bbl@xechars@#1\endcsname}}%
  \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}
```

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 a intermediate macro, which can be 'disabled' with `\bbl@ic@<label>@<lang>`.

```
\def\bbl@ignoreinterchar{%
  \ifnum\language=\l@nohyphenation
    \expandafter\@gobble
  \else
    \expandafter\@firstofone
  \fi}
\newcommand\babelinterchar[5][]{%
  \let\bbl@kv@label\@empty
  \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}%
  \@namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
    {\bbl@ignoreinterchar{#5}}%
  \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone
  \bbl@exp{\\\bbl@for\\\bbl@tempa{\zap@space#3 \@empty}}{%
    \bbl@exp{\\\bbl@for\\\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}{#1}{}{}}%
    {\bbl@csarg\let{ic@#1@\languagename}\@firstofone}}
\DeclareRobustCommand\disablelocaleinterchar[1]{%
  \bbl@ifunset{bbl@ic@#1@\languagename}%
    {\bbl@error{unknown-interchar-b}{#1}{}{}}%
    {\bbl@csarg\let{ic@#1@\languagename}\@gobble}}
⟨/xetex⟩
```

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.

```
⟨∗xetex | texxet⟩
\providecommand\bbl@provide@intraspace{}
\bbl@trace{Redefinitions for bidi layout}
\def\bbl@sspre@caption{%  TODO: Unused!
```

```
  \bbl@exp{\everyhbox{\\\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
\ifx\bbl@opt@layout\@nnil\else % if layout=..
\def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
\def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
\ifnum\bbl@bidimode>\z@  % TODO: always?
  \def\@hangfrom#1{%
    \setbox\@tempboxa\hbox{{#1}}%
    \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
    \noindent\box\@tempboxa}
  \def\raggedright{%
    \let\\\@centercr
    \bbl@startskip\z@skip
    \@rightskip\@flushglue
    \bbl@endskip\@rightskip
    \parindent\z@
    \parfillskip\bbl@startskip}
  \def\raggedleft{%
    \let\\\@centercr
    \bbl@startskip\@flushglue
    \bbl@endskip\z@skip
    \parindent\z@
    \parfillskip\bbl@endskip}
\fi
\IfBabelLayout{lists}
  {\bbl@sreplace\list
     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
   \def\bbl@listleftmargin{%
     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
   \ifcase\bbl@engine
     \def\labelenumii{)\theenumii(}% pdftex doesn't reverse ()
     \def\p@enumiii{\p@enumii)\theenumii(}%
   \fi
   \bbl@sreplace\@verbatim
     {\leftskip\@totalleftmargin}%
     {\bbl@startskip\textwidth
      \advance\bbl@startskip-\linewidth}%
   \bbl@sreplace\@verbatim
     {\rightskip\z@skip}%
     {\bbl@endskip\z@skip}}%
  {}
\IfBabelLayout{contents}
  {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
   \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
  {}
\IfBabelLayout{columns}
  {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputhbox}%
   \def\bbl@outputhbox#1{%
     \hb@xt@\textwidth{%
       \hskip\columnwidth
       \hfil
       {\normalcolor\vrule \@width\columnseprule}%
       \hfil
       \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
       \hskip-\textwidth
       \hb@xt@\columnwidth{\box\@outputbox \hss}%
       \hskip\columnsep
       \hskip\columnwidth}}%
  {}
⟨⟨Footnote changes⟩⟩
\IfBabelLayout{footnotes}%
  {\BabelFootnote\footnote\languagename{}{}%
   \BabelFootnote\localfootnote\languagename{}{}%
   \BabelFootnote\mainfootnote{}{}{}}
  {}
```

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*}%
  {\bbl@add\bbl@opt@layout{.counters.}%
   \AddToHook{shipout/before}{%
     \let\bbl@tempa\babelsublr
     \let\babelsublr\@firstofone
     \let\bbl@save@thepage\thepage
     \protected@edef\thepage{\thepage}%
     \let\babelsublr\bbl@tempa}%
   \AddToHook{shipout/after}{%
     \let\thepage\bbl@save@thepage}}{}
\IfBabelLayout{counters}%
  {\let\bbl@latinarabic=\@arabic
   \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
   \let\bbl@asciiroman=\@roman
   \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
   \let\bbl@asciiRoman=\@Roman
   \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
\fi % end if layout
⟨/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).

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

10.3 LuaTeX

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

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

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

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

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

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

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

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

```
⟨*luatex⟩
\ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
\bbl@trace{Read language.dat}
\ifx\bbl@readstream\@undefined
  \csname newread\endcsname\bbl@readstream
\fi
\begingroup
  \toks@{}
  \count@\z@ % 0=start, 1=0th, 2=normal
  \def\bbl@process@line#1#2 #3 #4 {%
    \ifx=#1%
      \bbl@process@synonym{#2}%
    \else
      \bbl@process@language{#1#2}{#3}{#4}%
    \fi
    \ignorespaces}
  \def\bbl@manylang{%
    \ifnum\bbl@last>\@ne
      \bbl@info{Non-standard hyphenation setup}%
    \fi
    \let\bbl@manylang\relax}
  \def\bbl@process@language#1#2#3{%
    \ifcase\count@
      \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
    \or
```

```
      \count@\tw@
    \fi
    \ifnum\count@=\tw@
      \expandafter\addlanguage\csname l@#1\endcsname
      \language\allocationnumber
      \chardef\bbl@last\allocationnumber
      \bbl@manylang
      \let\bbl@elt\relax
      \xdef\bbl@languages{%
        \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
    \fi
    \the\toks@
    \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}{}}
    \gdef\bbl@languages{%
      \bbl@elt{english}{0}{hyphen.tex}{}%
      \bbl@elt{USenglish}{0}{}{}}
  \else
    \global\let\bbl@languages@format\bbl@languages
    \def\bbl@elt#1#2#3#4{% Remove all except language 0
      \ifnum#2>\z@\else
        \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
      \fi}%
    \xdef\bbl@languages{\bbl@languages}%
  \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
    \repeat
  \fi
  \closein\bbl@readstream
\endgroup
\bbl@trace{Macros for reading patterns files}
```

```
\def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
\ifx\babelcatcodetablenum\@undefined
  \ifx\newcatcodetable\@undefined
    \def\babelcatcodetablenum{5211}
    \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
  \else
    \newcatcodetable\babelcatcodetablenum
    \newcatcodetable\bbl@pattcodes
  \fi
\else
  \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
\fi
\def\bbl@luapatterns#1#2{%
  \bbl@get@enc#1::\@@@
  \setbox\z@\hbox\bgroup
    \begingroup
      \savecatcodetable\babelcatcodetablenum\relax
      \initcatcodetable\bbl@pattcodes\relax
      \catcodetable\bbl@pattcodes\relax
        \catcode`\#=6  \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
        \catcode`\_=8  \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
        \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
        \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
        \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
        \catcode`\`=12 \catcode`\'=12 \catcode`\"=12
        \input #1\relax
      \catcodetable\babelcatcodetablenum\relax
    \endgroup
    \def\bbl@tempa{#2}%
    \ifx\bbl@tempa\@empty\else
      \input #2\relax
    \fi
  \egroup}%
\def\bbl@patterns@lua#1{%
  \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
    \csname l@#1\endcsname
    \edef\bbl@tempa{#1}%
  \else
    \csname l@#1:\f@encoding\endcsname
    \edef\bbl@tempa{#1:\f@encoding}%
  \fi\relax
  \@namedef{lu@texhyphen@loaded@\the\language}{}% Temp
  \@ifundefined{bbl@hyphendata@\the\language}%
    {\def\bbl@elt##1##2##3##4{%
       \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
         \def\bbl@tempb{##3}%
         \ifx\bbl@tempb\@empty\else % if not a synonymous
           \def\bbl@tempc{{##3}{##4}}%
         \fi
         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
       \fi}%
     \bbl@languages
     \@ifundefined{bbl@hyphendata@\the\language}%
       {\bbl@info{No hyphenation patterns were set for\\%
                  language '\bbl@tempa'. Reported}}%
       {\expandafter\expandafter\expandafter\bbl@luapatterns
          \csname bbl@hyphendata@\the\language\endcsname}}{}}
\endinput\fi
  % Here ends \ifx\AddBabelHook\@undefined
  % A few lines are only read by hyphen.cfg
\ifx\DisableBabelHook\@undefined
  \AddBabelHook{luatex}{everylanguage}{%
    \def\process@language##1##2##3{%
    \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{]] ..
      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.bytes)
    end
  end
  function Babel.end_process_input ()
    if luatexbase and luatexbase.remove_from_callback then
      luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
    else
      callback.register('process_input_buffer',Babel.callback)
    end
  end
  function Babel.addpatterns(pp, lg)
    local lg = lang.new(lg)
    local pats = lang.patterns(lg) or ''
    lang.clear_patterns(lg)
    for p in pp:gmatch('[^%s]+') do
      ss = ''
      for i in string.utfcharacters(p:gsub('%d', '')) do
         ss = ss .. '%d?' .. i
      end
      ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
      ss = ss:gsub('%.%%d%?$', '%%.')
      pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
      if n == 0 then
        tex.sprint(
          [[\string\csname\space bbl@info\endcsname{New pattern: ]]
          .. p .. [[}]])
        pats = pats .. ' ' .. p
      else
        tex.sprint(
          [[\string\csname\space bbl@info\endcsname{Renew pattern: ]]
          .. p .. [[}]])
      end
    end
    lang.patterns(lg, pats)
  end
  Babel.characters = Babel.characters or {}
  Babel.ranges = Babel.ranges or {}
  function Babel.hlist_has_bidi(head)
    local has_bidi = false
    local ranges = Babel.ranges
    for item in node.traverse(head) do
      if item.id == node.id'glyph' then
        local itemchar = item.char
        local chardata = Babel.characters[itemchar]
        local dir = chardata and chardata.d or nil
        if not dir then
          for nn, et in ipairs(ranges) do
            if itemchar < et[1] then
              break
            elseif itemchar <= et[2] then
              dir = et[3]
              break
            end
          end
        end
        if dir and (dir == 'al' or dir == 'r') then
          has_bidi = true
        end
      end
    end
    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( [[\string\indexentry]] ) and
         str:find( [[\string\babelsublr]] ) then
     str = str:gsub( [[\string\babelsublr%s*(%b{})]],
                     function(m) return m:sub(2,-2) end )
   end
   return str
end
}
\endgroup
\ifx\newattribute\@undefined\else % Test for plain
  \newattribute\bbl@attr@locale
  \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
  \AddBabelHook{luatex}{beforeextras}{%
    \setattribute\bbl@attr@locale\localeid}
\fi
\def\BabelStringsDefault{unicode}
\let\luabbl@stop\relax
\AddBabelHook{luatex}{encodedcommands}{%
```

```
  \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
  \ifx\bbl@tempa\bbl@tempb\else
    \directlua{Babel.begin_process_input()}%
    \def\luabbl@stop{%
      \directlua{Babel.end_process_input()}}%
  \fi}%
\AddBabelHook{luatex}{stopcommands}{%
  \luabbl@stop
  \let\luabbl@stop\relax}
\AddBabelHook{luatex}{patterns}{%
  \@ifundefined{bbl@hyphendata@\the\language}%
    {\def\bbl@elt##1##2##3##4{%
       \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
         \def\bbl@tempb{##3}%
         \ifx\bbl@tempb\@empty\else % if not a synonymous
           \def\bbl@tempc{{##3}{##4}}%
         \fi
         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
       \fi}%
     \bbl@languages
     \@ifundefined{bbl@hyphendata@\the\language}%
       {\bbl@info{No hyphenation patterns were set for\\%
                  language '#2'. Reported}}%
       {\expandafter\expandafter\expandafter\bbl@luapatterns
          \csname bbl@hyphendata@\the\language\endcsname}}{}%
  \@ifundefined{bbl@patterns@}{}{%
    \begingroup
      \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
      \ifin@\else
        \ifx\bbl@patterns@\@empty\else
           \directlua{ Babel.addpatterns(
             [[\bbl@patterns@]], \number\language) }%
        \fi
        \@ifundefined{bbl@patterns@#1}%
          \@empty
          {\directlua{ Babel.addpatterns(
               [[\space\csname bbl@patterns@#1\endcsname]],
               \number\language) }}%
        \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
      \fi
    \endgroup}%
  \bbl@exp{%
    \bbl@ifunset{bbl@prehc@\languagename}{}%
      {\\\bbl@ifblank{\bbl@cs{prehc@\languagename}}{}%
        {\prehyphenchar=\bbl@cl{prehc}\relax}}}}
```

`\babelpatterns` This macro adds patterns. Two macros are used to store them: `\bbl@patterns@` for the global ones and `\bbl@patterns@<lang>` for language ones. We make sure there is a space between words when multiple commands are used.

```
\@onlypreamble\babelpatterns
\AtEndOfPackage{%
  \newcommand\babelpatterns[2][\@empty]{%
    \ifx\bbl@patterns@\relax
      \let\bbl@patterns@\@empty
    \fi
    \ifx\bbl@pttnlist\@empty\else
      \bbl@warning{%
        You must not intermingle \string\selectlanguage\space and\\%
        \string\babelpatterns\space or some patterns will not\\%
        be taken into account. Reported}%
    \fi
    \ifx\@empty#1%
      \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
    \else
      \edef\bbl@tempb{\zap@space#1 \@empty}%
      \bbl@for\bbl@tempa\bbl@tempb{%
        \bbl@fixname\bbl@tempa
        \bbl@iflanguage\bbl@tempa{%
          \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
            \@ifundefined{bbl@patterns@\bbl@tempa}%
              \@empty
              {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
            #2}}}%
    \fi}}
```

10.4 Southeast Asian scripts

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

```
% TODO - to a lua file
\directlua{
  Babel = Babel or {}
  Babel.linebreaking = Babel.linebreaking or {}
  Babel.linebreaking.before = {}
  Babel.linebreaking.after = {}
  Babel.locale = {} % Free to use, indexed by \localeid
  function Babel.linebreaking.add_before(func, pos)
    tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
    if pos == nil then
      table.insert(Babel.linebreaking.before, func)
    else
      table.insert(Babel.linebreaking.before, pos, func)
    end
  end
  function Babel.linebreaking.add_after(func)
    tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
    table.insert(Babel.linebreaking.after, func)
  end
}
\def\bbl@intraspace#1 #2 #3\@@{%
  \directlua{
    Babel = Babel or {}
    Babel.intraspaces = Babel.intraspaces or {}
    Babel.intraspaces['\csname bbl@sbcp@\languagename\endcsname'] = %
       {b = #1, p = #2, m = #3}
    Babel.locale_props[\the\localeid].intraspace = %
       {b = #1, p = #2, m = #3}
  }}
\def\bbl@intrapenalty#1\@@{%
  \directlua{
    Babel = Babel or {}
    Babel.intrapenalties = Babel.intrapenalties or {}
    Babel.intrapenalties['\csname bbl@sbcp@\languagename\endcsname'] = #1
    Babel.locale_props[\the\localeid].intrapenalty = #1
  }}
\begingroup
\catcode`\%=12
\catcode`\&=14
\catcode`\'=12
\catcode`\~=12
\gdef\bbl@seaintraspace{&
  \let\bbl@seaintraspace\relax
  \directlua{
    Babel = Babel or {}
    Babel.sea_enabled = true
    Babel.sea_ranges = Babel.sea_ranges or {}
    function Babel.set_chranges (script, chrng)
      local c = 0
      for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
        Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
        c = c + 1
      end
    end
    function Babel.sea_disc_to_space (head)
      local sea_ranges = Babel.sea_ranges
      local last_char = nil
      local quad = 655360      &% 10 pt = 655360 = 10 * 65536
      for item in node.traverse(head) do
        local i = item.id
        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
  }&
  \bbl@luahyphenate}
```

10.5 CJK line breaking

Minimal line breaking for CJK scripts, mainly intended for simple documents and short texts as a secondary language. Only line breaking, with a little stretching for justification, without any attempt to adjust the spacing. It is based on (but does not strictly follow) the Unicode algorithm.
We first need a little table with the corresponding line breaking properties. A few characters have an additional key for the width (fullwidth *vs.* halfwidth), not yet used. There is a separate file, defined below.

```
\catcode`\%=14
\gdef\bbl@cjkintraspace{%
  \let\bbl@cjkintraspace\relax
  \directlua{
    Babel = Babel or {}
    require('babel-data-cjk.lua')
    Babel.cjk_enabled = true
    function Babel.cjk_linebreak(head)
      local GLYPH = node.id'glyph'
      local last_char = nil
      local quad = 655360      % 10 pt = 655360 = 10 * 65536
      local last_class = nil
      local last_lang = nil

      for item in node.traverse(head) do
        if item.id == GLYPH then

          local lang = item.lang

          local LOCALE = node.get_attribute(item,
                Babel.attr_locale)
          local props = Babel.locale_props[LOCALE]

          local class = Babel.cjk_class[item.char].c

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

          if class == 'cp' then class = 'cl' end % )] as CL
          if class == 'id' then class = 'I' end

          local br = 0
          if class and last_class and Babel.cjk_breaks[last_class][class] then
            br = Babel.cjk_breaks[last_class][class]
          end

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

          if font.getfont(item.font) then
            quad = font.getfont(item.font).size
          end
          last_class = class
          last_lang = lang
        else % if penalty, glue or anything else
          last_class = nil
        end
      end
      lang.hyphenate(head)
    end
  }%
  \bbl@luahyphenate}
\gdef\bbl@luahyphenate{%
  \let\bbl@luahyphenate\relax
  \directlua{
    luatexbase.add_to_callback('hyphenate',
    function (head, tail)
      if Babel.linebreaking.before then
        for k, func in ipairs(Babel.linebreaking.before)  do
          func(head)
        end
      end
      if Babel.cjk_enabled then
        Babel.cjk_linebreak(head)
      end
      lang.hyphenate(head)
      if Babel.linebreaking.after then
        for k, func in ipairs(Babel.linebreaking.after)  do
          func(head)
        end
      end
      if Babel.sea_enabled then
        Babel.sea_disc_to_space(head)
      end
    end,
    'Babel.hyphenate')
  }
}
\endgroup
\def\bbl@provide@intraspace{%
  \bbl@ifunset{bbl@intsp@\languagename}{}%
    {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
       \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
       \ifin@          % cjk
         \bbl@cjkintraspace
         \directlua{
             Babel = Babel or {}
             Babel.locale_props = Babel.locale_props or {}
             Babel.locale_props[\the\localeid].linebreak = 'c'
         }%
         \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
         \ifx\bbl@KVP@intrapenalty\@nnil
           \bbl@intrapenalty0\@@
         \fi
       \else           % sea
         \bbl@seaintraspace
         \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
         \directlua{
            Babel = Babel or {}
            Babel.sea_ranges = Babel.sea_ranges or {}
            Babel.set_chranges('\bbl@cl{sbcp}',
                               '\bbl@cl{chrng}')
         }%
         \ifx\bbl@KVP@intrapenalty\@nnil
           \bbl@intrapenalty0\@@
         \fi
       \fi
     \fi
     \ifx\bbl@KVP@intrapenalty\@nnil\else
       \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
     \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-`

```
\ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
\def\bblar@chars{%
  0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
  0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
  0640,0641,0642,0643,0644,0645,0646,0647,0649}
```

```
\def\bblar@elongated{%
  0626,0628,062A,062B,0633,0634,0635,0636,063B,%
  063C,063D,063E,063F,0641,0642,0643,0644,0646,%
  0649,064A}
\begingroup
  \catcode`_=11 \catcode`:=11
  \gdef\bblar@nofswarn{\gdef\msg_warning:nnx##1##2##3{}}
\endgroup
\gdef\bbl@arabicjust{% TODO. Allow for several locales.
  \let\bbl@arabicjust\relax
  \newattribute\bblar@kashida
  \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
  \bblar@kashida=\z@
  \bbl@patchfont{{\bbl@parsejalt}}%
  \directlua{
    Babel.arabic.elong_map   = Babel.arabic.elong_map or {}
    Babel.arabic.elong_map[\the\localeid]   = {}
    luatexbase.add_to_callback('post_linebreak_filter',
      Babel.arabic.justify, 'Babel.arabic.justify')
    luatexbase.add_to_callback('hpack_filter',
      Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
  }}%
```

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

```
\def\bblar@fetchjalt#1#2#3#4{%
  \bbl@exp{\\\bbl@foreach{#1}}{%
    \bbl@ifunset{bblar@JE@##1}%
      {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"##1#2}}%
      {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"\@nameuse{bblar@JE@##1}#2}}%
    \directlua{%
      local last = nil
      for item in node.traverse(tex.box[0].head) do
        if item.id == node.id'glyph' and item.char > 0x600 and
            not (item.char == 0x200D) then
          last = item
        end
      end
      Babel.arabic.#3['##1#4'] = last.char
    }}}
```

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

```
\gdef\bbl@parsejalt{%
  \ifx\addfontfeature\@undefined\else
    \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
    \ifin@
      \directlua{%
        if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
          Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
          tex.print([[\string\csname\space bbl@parsejalti\endcsname]])
        end
      }%
    \fi
  \fi}
\gdef\bbl@parsejalti{%
  \begingroup
    \let\bbl@parsejalt\relax     % To avoid infinite loop
    \edef\bbl@tempb{\fontid\font}%
    \bblar@nofswarn
    \bblar@fetchjalt\bblar@elongated{}{from}{}%
    \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
    \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
    \addfontfeature{RawFeature=+jalt}%
    % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
    \bblar@fetchjalt\bblar@elongated{}{dest}{}%
    \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
    \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
      \directlua{%
        for k, v in pairs(Babel.arabic.from) do
          if Babel.arabic.dest[k] and
              not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
            Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
               [Babel.arabic.from[k]] = Babel.arabic.dest[k]
          end
        end
      }%
  \endgroup}
```

The actual justification (inspired by CHICKENIZE).

```
\begingroup
\catcode`#=11
\catcode`~=11
\directlua{

Babel.arabic = Babel.arabic or {}
Babel.arabic.from = {}
Babel.arabic.dest = {}
Babel.arabic.justify_factor = 0.95
Babel.arabic.justify_enabled = true
Babel.arabic.kashida_limit = -1

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

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

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

  if line == nil then
    line = {}
    line.glue_sign = 1
    line.glue_order = 0
    line.head = head
    line.shift = 0
    line.width = size
  end

  % 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

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

    end % of node.traverse_id

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

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

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

```

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

    while width < goal do
      subst_done = true
      k_item = k_list[k_curr].node
      if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
        d = node.copy(k_item)
        d.char = 0x0640
        d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
        d.xoffset = 0
        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
        else
          cnt = 0
        end
      end
    end

    ::next_line::

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

10.7 Common stuff

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

10.8 Automatic fonts and ids switching

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

```
% TODO - to a lua file
\directlua{
Babel.script_blocks = {
  ['dflt'] = {},
  ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
              {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
  ['Armn'] = {{0x0530, 0x058F}},
  ['Beng'] = {{0x0980, 0x09FF}},
  ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
  ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
  ['Cyrl'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
  ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
  ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
              {0xAB00, 0xAB2F}},
  ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
  % Don't follow strictly Unicode, which places some Coptic letters in
  % the 'Greek and Coptic' block
  ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
  ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
              {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
              {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
              {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
              {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
              {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
  ['Hebr'] = {{0x0590, 0x05FF}},
  ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
              {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
  ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
  ['Knda'] = {{0x0C80, 0x0CFF}},
  ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
              {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
              {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
  ['Laoo'] = {{0x0E80, 0x0EFF}},
  ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
              {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
              {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
  ['Mahj'] = {{0x11150, 0x1117F}},
  ['Mlym'] = {{0x0D00, 0x0D7F}},
  ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
  ['Orya'] = {{0x0B00, 0x0B7F}},
  ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
  ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
  ['Taml'] = {{0x0B80, 0x0BFF}},
  ['Telu'] = {{0x0C00, 0x0C7F}},
  ['Tfng'] = {{0x2D30, 0x2D7F}},
  ['Thai'] = {{0x0E00, 0x0E7F}},
  ['Tibt'] = {{0x0F00, 0x0FFF}},
  ['Vaii'] = {{0xA500, 0xA63F}},
  ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
}

Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
Babel.script_blocks.Hant = Babel.script_blocks.Hans
Babel.script_blocks.Kana = Babel.script_blocks.Jpan

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
        % Treat composite chars in a different fashion, because they
        % 'inherit' the previous locale.
        if (item.char >= 0x0300 and item.char <= 0x036F) or
           (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
           (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
             Babel.chr_to_loc[item.char] = -2000
             toloc = -2000
        end
        if not toloc then
          Babel.chr_to_loc[item.char] = -1000
        end
      end
      if toloc == -2000 then
        toloc = toloc_save
      elseif toloc == -1000 then
        toloc = nil
      end
      if toloc and Babel.locale_props[toloc] and
          Babel.locale_props[toloc].letters and
          tex.getcatcode(item.char) \string~= 11 then
        toloc = nil
      end
      if toloc and Babel.locale_props[toloc].script
          and Babel.locale_props[node.get_attribute(item, LOCALE)].script
          and Babel.locale_props[toloc].script ==
            Babel.locale_props[node.get_attribute(item, LOCALE)].script then
        toloc = nil
      end
      if toloc then
        if Babel.locale_props[toloc].lg then
          item.lang = Babel.locale_props[toloc].lg
          node.set_attribute(item, LOCALE, toloc)
        end
        if Babel.locale_props[toloc]['/'..item.font] then
          item.font = Babel.locale_props[toloc]['/'..item.font]
        end
      end
      toloc_save = toloc
    elseif not inmath and item.id == 7 then % Apply recursively
      item.replace = item.replace and Babel.locale_map(item.replace)
      item.pre     = item.pre and Babel.locale_map(item.pre)
      item.post    = item.post and Babel.locale_map(item.post)
    elseif item.id == node.id'math' then
      inmath = (item.subtype == 0)
    end
  end
  return head
end
}
```

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

```
\newcommand\babelcharproperty[1]{%
  \count@=#1\relax
  \ifvmode
    \expandafter\bbl@chprop
  \else
    \bbl@error{charproperty-only-vertical}{}{}{}%
  \fi}
\newcommand\bbl@chprop[3][\the\count@]{%
  \@tempcnta=#1\relax
  \bbl@ifunset{bbl@chprop@#2}% {unknown-char-property}
    {\bbl@error{unknown-char-property}{}{#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}}\space
  }}
```

Post-handling hyphenation patterns for non-standard rules, like `ff` to `ff-f`. There are still some issues with speed (not very slow, but still slow). The Lua code is below.

```
\directlua{
  Babel.nohyphenation = \the\l@nohyphenation
}
```

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

```
\begingroup
\catcode`\~=12
\catcode`\%=12
\catcode`\&=14
\catcode`\|=12
\gdef\babelprehyphenation{&%
  \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}}
\gdef\babelposthyphenation{&%
  \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}}
\gdef\bbl@settransform#1[#2]#3#4#5{&%
  \ifcase#1
    \bbl@activateprehyphen
  \or
    \bbl@activateposthyphen
  \fi
  \begingroup
    \def\babeltempa{\bbl@add@list\babeltempb}&%
    \let\babeltempb\@empty
    \def\bbl@tempa{#5}&%
    \bbl@replace\bbl@tempa{,}{ ,}&% TODO. Ugly trick to preserve {}
    \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
      \bbl@ifsamestring{##1}{remove}&%
        {\bbl@add@list\babeltempb{nil}}&%
        {\directlua{
           local rep = [=[##1]=]
           rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
           rep = rep:gsub('^%s*(insert)%s*,', 'insert = true, ')
           rep = rep:gsub('^%s*(after)%s*,', 'after = true, ')
           rep = rep:gsub('(string)%s*=%s*([^%s,]*)', Babel.capture_func)
           rep = rep:gsub('node%s*=%s*(%a+)%s*(%a*)', Babel.capture_node)
           rep = rep:gsub(&%
             '(norule)%s*=%s*([%-%d%.]+)%s+([%-%d%.]+)%s+([%-%d%.]+)',
             'norule = {' .. '%2, %3, %4' .. '}')
           if #1 == 0 or #1 == 2 then
             rep = rep:gsub(&%
               '(space)%s*=%s*([%-%d%.]+)%s+([%-%d%.]+)%s+([%-%d%.]+)',
               'space = {' .. '%2, %3, %4' .. '}')
             rep = rep:gsub(&%
               '(spacefactor)%s*=%s*([%-%d%.]+)%s+([%-%d%.]+)%s+([%-%d%.]+)',
               'spacefactor = {' .. '%2, %3, %4' .. '}')
             rep = rep:gsub('(kashida)%s*=%s*([^%s,]*)', Babel.capture_kashida)
           else
             rep = rep:gsub(    '(no)%s*=%s*([^%s,]*)', Babel.capture_func)
             rep = rep:gsub(   '(pre)%s*=%s*([^%s,]*)', Babel.capture_func)
             rep = rep:gsub(  '(post)%s*=%s*([^%s,]*)', Babel.capture_func)
           end
           tex.print([[\string\babeltempa{{]] .. rep .. [[}}]])
         }}}&%
    \bbl@foreach\babeltempb{&%
      \bbl@forkv{{##1}}{&%
        \in@{,####1,}{,nil,step,data,remove,insert,string,no,pre,no,&%
          post,penalty,kashida,space,spacefactor,kern,node,after,norule,}&%
        \ifin@\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
        \bbl@exp{\\\bbl@trim@def\\\bbl@kv@fonts{\bbl@kv@fonts}}&%
        \bbl@replace\bbl@kv@fonts{ }{,}&%
        \edef\bbl@kv@attribute{bbl@ATR@\bbl@kv@label @#3@\bbl@kv@fonts}&%
        \count@\z@
        \def\bbl@elt##1##2##3{&%
          \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
            {\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}}&%
          {}&%
        \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
      \fi
    \else
      \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
    \fi
    \directlua{
      local lbkr = Babel.linebreaking.replacements[#1]
      local u = unicode.utf8
      local id, attr, label
      if #1 == 0 then
        id = \the\csname bbl@id@@#3\endcsname\space
      else
        id = \the\csname l@#3\endcsname\space
      end
      \ifx\bbl@kv@attribute\relax
        attr = -1
      \else
        attr = luatexbase.registernumber'\bbl@kv@attribute'
      \fi
      \ifx\bbl@kv@label\relax\else  &% Same refs:
        label = [==[\bbl@kv@label]==]
      \fi
      &% Convert pattern:
      local patt = string.gsub([==[#4]==], '%s', '')
      if #1 == 0 then
        patt = string.gsub(patt, '|', ' ')
      end
      if not u.find(patt, '()', nil, true) then
        patt = '()' .. patt .. '()'
      end
      if #1 == 1 then
        patt = string.gsub(patt, '%(%)%^', '^()')
```

```
        patt = string.gsub(patt, '%$%(%)', '()$')
      end
      patt = u.gsub(patt, '{(.)}',
             function (n)
               return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
             end)
      patt = u.gsub(patt, '{(%x%x%x%x+)}',
             function (n)
               return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
             end)
      lbkr[id] = lbkr[id] or {}
      table.insert(lbkr[id],
        { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
    }&%
  \endgroup}
\endgroup
\let\bbl@transfont@list\@empty
\def\bbl@settransfont{%
  \global\let\bbl@settransfont\relax % Execute only once
  \gdef\bbl@transfont{%
    \def\bbl@elt####1####2####3{%
      \bbl@ifblank{####3}%
         {\count@\tw@}% Do nothing if no fonts
         {\count@\z@
          \bbl@vforeach{####3}{%
            \def\bbl@tempd{########1}%
            \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
            \ifx\bbl@tempd\bbl@tempe
              \count@\@ne
            \else\ifx\bbl@tempd\bbl@transfam
              \count@\@ne
            \fi\fi}%
         \ifcase\count@
           \bbl@csarg\unsetattribute{ATR@####2@####1@####3}%
         \or
           \bbl@csarg\setattribute{ATR@####2@####1@####3}\@ne
         \fi}}%
      \bbl@transfont@list}%
  \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
  \gdef\bbl@transfam{-unknown-}%
  \bbl@foreach\bbl@font@fams{%
    \AddToHook{##1family}{\def\bbl@transfam{##1}}%
    \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
      {\xdef\bbl@transfam{##1}}%
      {}}}
\DeclareRobustCommand\enablelocaletransform[1]{%
  \bbl@ifunset{bbl@ATR@#1@\languagename @}%
    {\bbl@error{transform-not-available}{#1}{}{}}%
    {\bbl@csarg\setattribute{ATR@#1@\languagename @}\@ne}}
\DeclareRobustCommand\disablelocaletransform[1]{%
  \bbl@ifunset{bbl@ATR@#1@\languagename @}%
    {\bbl@error{transform-not-available-b}{#1}{}{}}%
    {\bbl@csarg\unsetattribute{ATR@#1@\languagename @}}}
\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')
    Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
  }}
```

The following experimental (and unfinished) macro applies the prehyphenation transforms for the current locale to a string (characters and spaces) and processes it in a fully expandable way (among other limitations, the string can't contain `]==]`). The way it operates is admittedly rather cumbersome: it converts the string to a node list, processes it, and converts it back to a string. The lua code is in the lua file below.

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

10.9 Bidi

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

```
\def\bbl@activate@preotf{%
  \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) %%% TODO
      if Babel.numbers and Babel.digits_mapped then
        head = Babel.numbers(head)
      end
      if Babel.bidi_enabled then
        head = Babel.bidi(head, false, dir)
      end
      return head
    end
    %
    luatexbase.add_to_callback('pre_linebreak_filter',
      Babel.pre_otfload_v,
      'Babel.pre_otfload_v',
      luatexbase.priority_in_callback('pre_linebreak_filter',
        'luaotfload.node_processor') or nil)
    %
    luatexbase.add_to_callback('hpack_filter',
      Babel.pre_otfload_h,
      'Babel.pre_otfload_h',
      luatexbase.priority_in_callback('hpack_filter',
        'luaotfload.node_processor') or nil)
  }}
```

The basic setup. The output is modified at a very low level to set the `\bodydir` to the `\pagedir`. Sadly, we have to deal with boxes in math with basic, so the `\bbl@mathboxdir` hack is activated every math with the package option bidi=. The hack for the PUA is no longer necessary with `basic`, but it's kept in `basic-r`.

```
\breakafterdirmode=1
\ifnum\bbl@bidimode>\@ne % Any bidi= except default=1
  \let\bbl@beforeforeign\leavevmode
  \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
  \RequirePackage{luatexbase}
  \bbl@activate@preotf
  \directlua{
    require('babel-data-bidi.lua')
    \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
      require('babel-bidi-basic.lua')
    \or
      require('babel-bidi-basic-r.lua')
      table.insert(Babel.ranges, {0xE000,   0xF8FF, 'on'})
      table.insert(Babel.ranges, {0xF0000,  0xFFFFD, 'on'})
      table.insert(Babel.ranges, {0x100000, 0x10FFFD, 'on'})
    \fi}
  \newattribute\bbl@attr@dir
  \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
  \bbl@exp{\output{\bodydir\pagedir\the\output}}
\fi
\chardef\bbl@thetextdir\z@
\chardef\bbl@thepardir\z@
\def\bbl@getluadir#1{%
  \directlua{
    if tex.#1dir == 'TLT' then
      tex.sprint('0')
    elseif tex.#1dir == 'TRT' then
      tex.sprint('1')
    end}}
\def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
  \ifcase#3\relax
    \ifcase\bbl@getluadir{#1}\relax\else
      #2 TLT\relax
    \fi
  \else
    \ifcase\bbl@getluadir{#1}\relax
      #2 TRT\relax
    \fi
  \fi}
% ..00PPTT, with masks 0xC (par dir) and 0x3 (text dir)
\def\bbl@thedir{0}
\def\bbl@textdir#1{%
  \bbl@setluadir{text}\textdir{#1}%
  \chardef\bbl@thetextdir#1\relax
  \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
  \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
\def\bbl@pardir#1{%  Used twice
  \bbl@setluadir{par}\pardir{#1}%
  \chardef\bbl@thepardir#1\relax}
\def\bbl@bodydir{\bbl@setluadir{body}\bodydir}%   Used once
\def\bbl@pagedir{\bbl@setluadir{page}\pagedir}%   Unused
\def\bbl@dirparastext{\pardir\the\textdir\relax}% Used once
```

RTL text inside math needs special attention. It affects not only to actual math stuff, but also to ‘tabular‘, which is based on a fake math.

```
\ifnum\bbl@bidimode>\z@ % Any bidi=
  \def\bbl@insidemath{0}%
  \def\bbl@everymath{\def\bbl@insidemath{1}}
  \def\bbl@everydisplay{\def\bbl@insidemath{2}}
  \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)
            local inmath = false
            for item in node.traverse(head) do
              if item.id == 11 then
                inmath = (item.subtype == 0)
              elseif not inmath then
                node.set_attribute(item,
                  Babel.attr_dir, token.get_macro('bbl@thedir'))
              end
            end
          end
        end
        return head
      end
      luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
        "Babel.math_box_dir", 0)
      if Babel.unset_atdir then
        luatexbase.add_to_callback("pre_linebreak_filter", Babel.unset_atdir,
          "Babel.unset_atdir")
        luatexbase.add_to_callback("hpack_filter", Babel.unset_atdir,
          "Babel.unset_atdir")
      end
  }}%
\fi
```

Experimental. Tentative name.

```
\DeclareRobustCommand\localebox[1]{%
  {\def\bbl@insidemath{0}%
   \mbox{\foreignlanguage{\languagename}{#1}}}}
```

10.10 Layout

Unlike xetex, luatex requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with bidi=basic, without having to patch almost any macro where text direction is relevant.
Still, there are three areas deserving special attention, namely, tabular, math, and graphics, text and intrinsically left-to-right elements are intermingled. I've made some progress in graphics, but they're essentially hacks; I've also made some progress in 'tabular', but when I decided to tackle math (both standard math and 'amsmath') the nightmare began. I'm still not sure how 'amsmath' should be modified, but the main problem is that, boxes are "generic" containers that can hold text, math, and graphics (even at the same time; remember that inline math is included in the list of text nodes marked with 'math' (11) nodes too).
\@hangfrom is useful in many contexts and it is redefined always with the layout option.
There are, however, a number of issues when the text direction is not the same as the box direction (as set by \bodydir), and when \parbox and \hangindent are involved. Fortunately, latest releases of luatex simplify a lot the solution with \shapemode.
With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, tabular seems to work (at least in simple cases) with array, tabularx, hhline, colortbl, longtable, booktabs, etc. However, dcolumn still fails.

```
\bbl@trace{Redefinitions for bidi layout}
%
⟨⟨*More package options⟩⟩ ≡
\chardef\bbl@eqnpos\z@
\DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
\DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
⟨⟨/More package options⟩⟩
%
\ifnum\bbl@bidimode>\z@ % Any bidi=
  \matheqdirmode\@ne % A luatex primitive
```

```
\let\bbl@eqnodir\relax
\def\bbl@eqdel{()}
\def\bbl@eqnum{%
  {\normalfont\normalcolor
   \expandafter\@firstoftwo\bbl@eqdel
   \theequation
   \expandafter\@secondoftwo\bbl@eqdel}}
\def\bbl@puteqno#1{\eqno\hbox{#1}}
\def\bbl@putleqno#1{\leqno\hbox{#1}}
\def\bbl@eqno@flip#1{%
  \ifdim\predisplaysize=-\maxdimen
    \eqno
    \hb@xt@.01pt{%
      \hb@xt@\displaywidth{\hss{#1\glet\bbl@upset\@currentlabel}}\hss}%
  \else
    \leqno\hbox{#1\glet\bbl@upset\@currentlabel}%
  \fi
  \bbl@exp{\def\\\@currentlabel{\[bbl@upset]}}}
\def\bbl@leqno@flip#1{%
  \ifdim\predisplaysize=-\maxdimen
    \leqno
    \hb@xt@.01pt{%
      \hss\hb@xt@\displaywidth{{#1\glet\bbl@upset\@currentlabel}\hss}}%
  \else
    \eqno\hbox{#1\glet\bbl@upset\@currentlabel}%
  \fi
  \bbl@exp{\def\\\@currentlabel{\[bbl@upset]}}}
\AtBeginDocument{%
  \ifx\bbl@noamsmath\relax\else
  \ifx\maketag@@@\@undefined % Normal equation, eqnarray
    \AddToHook{env/equation/begin}{%
      \ifnum\bbl@thetextdir>\z@
        \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
        \let\@eqnnum\bbl@eqnum
        \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
        \chardef\bbl@thetextdir\z@
        \bbl@add\normalfont{\bbl@eqnodir}%
        \ifcase\bbl@eqnpos
          \let\bbl@puteqno\bbl@eqno@flip
        \or
          \let\bbl@puteqno\bbl@leqno@flip
        \fi
      \fi}%
    \ifnum\bbl@eqnpos=\tw@\else
      \def\endequation{\bbl@puteqno{\@eqnnum}$$\@ignoretrue}%
    \fi
    \AddToHook{env/eqnarray/begin}{%
      \ifnum\bbl@thetextdir>\z@
        \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
        \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
        \chardef\bbl@thetextdir\z@
        \bbl@add\normalfont{\bbl@eqnodir}%
        \ifnum\bbl@eqnpos=\@ne
          \def\@eqnnum{%
            \setbox\z@\hbox{\bbl@eqnum}%
            \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
        \else
          \let\@eqnnum\bbl@eqnum
        \fi
      \fi}
    % Hack. YA luatex bug?:
    \expandafter\bbl@sreplace\csname] \endcsname{$$}{\eqno\kern.001pt$$}%
  \else % amstex
```

```
    \bbl@exp{% Hack to hide maybe undefined conditionals:
      \chardef\bbl@eqnpos=0%
        \<iftagsleft@>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
    \ifnum\bbl@eqnpos=\@ne
      \let\bbl@ams@lap\hbox
    \else
      \let\bbl@ams@lap\llap
    \fi
    \ExplSyntaxOn % Required by \bbl@sreplace with \intertext@
    \bbl@sreplace\intertext@{\normalbaselines}%
      {\normalbaselines
       \ifx\bbl@eqnodir\relax\else\bbl@pardir\@ne\bbl@eqnodir\fi}%
    \ExplSyntaxOff
    \def\bbl@ams@tagbox#1#2{#1{\bbl@eqnodir#2}}% #1=hbox|@lap|flip
    \ifx\bbl@ams@lap\hbox % leqno
      \def\bbl@ams@flip#1{%
        \hbox to 0.01pt{\hss\hbox to\displaywidth{{#1}\hss}}}%
    \else % eqno
      \def\bbl@ams@flip#1{%
        \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}}\hss}}%
    \fi
    \def\bbl@ams@preset#1{%
      \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
      \ifnum\bbl@thetextdir>\z@
        \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
        \bbl@sreplace\textdef@{\hbox}{\bbl@ams@tagbox\hbox}%
        \bbl@sreplace\maketag@@@{\hbox}{\bbl@ams@tagbox#1}%
      \fi}%
    \ifnum\bbl@eqnpos=\tw@\else
      \def\bbl@ams@equation{%
        \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
        \ifnum\bbl@thetextdir>\z@
          \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
          \chardef\bbl@thetextdir\z@
          \bbl@add\normalfont{\bbl@eqnodir}%
          \ifcase\bbl@eqnpos
            \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
          \or
            \def\veqno##1##2{\bbl@leqno@flip{##1##2}}%
          \fi
        \fi}%
      \AddToHook{env/equation/begin}{\bbl@ams@equation}%
      \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
    \fi
    \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
    \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
    \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
    \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
    \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
    \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
    \AddToHook{env/alignat/begin}{\bbl@ams@preset\bbl@ams@lap}%
    \AddToHook{env/alignat*/begin}{\bbl@ams@preset\bbl@ams@lap}%
    \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
    % Hackish, for proper alignment. Don’t ask me why it works!:
    \bbl@exp{% Avoid a 'visible' conditional
      \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\\tag*{}\<fi>}%
      \\\AddToHook{env/alignat*/end}{\<iftag@>\<else>\\\tag*{}\<fi>}}%
    \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
    \AddToHook{env/split/before}{%
      \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
      \ifnum\bbl@thetextdir>\z@
        \bbl@ifsamestring\@currenvir{equation}%
          {\ifx\bbl@ams@lap\hbox % leqno
```

```
            \def\bbl@ams@flip#1{%
              \hbox to 0.01pt{\hbox to\displaywidth{{#1}\hss}\hss}}%
          \else
            \def\bbl@ams@flip#1{%
              \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}}%
          \fi}%
        {}%
     \fi}%
  \fi\fi}
\fi
\def\bbl@provide@extra#1{%
  % == Counters: mapdigits ==
  % Native digits
  \ifx\bbl@KVP@mapdigits\@nnil\else
    \bbl@ifunset{bbl@dgnat@\languagename}{}%
      {\RequirePackage{luatexbase}%
       \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
             return head
           end
         end
      }}%
  \fi
  % == transforms ==
  \ifx\bbl@KVP@transforms\@nnil\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}%
    \bbl@exp{%
      \\\bbl@ifblank{\bbl@cl{dgnat}}%
       {\let\\\bbl@tempa\relax}%
       {\def\\\bbl@tempa{%
         \\\bbl@elt{transforms.prehyphenation}%
          {digits.native.1.0}{([0-9])}%
         \\\bbl@elt{transforms.prehyphenation}%
          {digits.native.1.1}{string={1|0123456789|\bbl@cl{dgnat}}}}}}%
    \ifx\bbl@tempa\relax\else
      \bbl@exp{%
        \def\<bbl@inidata@\languagename>{%
          \[bbl@tempa]\[bbl@inidata@\languagename]}}%
    \fi
    \csname bbl@inidata@\languagename\endcsname
    \bbl@release@transforms\relax % \relax closes the last item.
  \fi}
% Start tabular here:
\def\localerestoredirs{%
  \ifcase\bbl@thetextdir
    \ifnum\textdirection=\z@\else\textdir TLT\fi
  \else
    \ifnum\textdirection=\@ne\else\textdir TRT\fi
  \fi
  \ifcase\bbl@thepardir
    \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
  \else
    \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
  \fi}
\IfBabelLayout{tabular}%
  {\chardef\bbl@tabular@mode\tw@}% All RTL
  {\IfBabelLayout{notabular}%
    {\chardef\bbl@tabular@mode\z@}%
    {\chardef\bbl@tabular@mode\@ne}}% Mixed, with LTR cols
\ifnum\bbl@bidimode>\@ne % Any lua bidi= except default=1
  % Redefine: vrules mess up dirs:
  \def\@arstrut{\relax\copy\@arstrutbox}%
  \ifcase\bbl@tabular@mode\or % 1 = Mixed - default
    \let\bbl@parabefore\relax
    \AddToHook{para/before}{\bbl@parabefore}
    \AtBeginDocument{%
      \bbl@replace\@tabular{$}{$%
        \def\bbl@insidemath{0}%
        \def\bbl@parabefore{\localerestoredirs}}%
      \ifnum\bbl@tabular@mode=\@ne
        \bbl@ifunset{@tabclassz}{}{%
          \bbl@exp{% Hide conditionals
            \\\bbl@sreplace\\\@tabclassz
              {\<ifcase>\\\@chnum}%
              {\\\localerestoredirs\<ifcase>\\\@chnum}}}%
        \@ifpackageloaded{colortbl}%
          {\bbl@sreplace\@classz
            {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
          {\@ifpackageloaded{array}%
             {\bbl@exp{% Hide conditionals
                \\\bbl@sreplace\\\@classz
                  {\<ifcase>\\\@chnum}%
                  {\bgroup\\\localerestoredirs\<ifcase>\\\@chnum}%
                \\\bbl@sreplace\\\@classz
                  {\\\do@row@strut\<fi>}{\\\do@row@strut\<fi>\egroup}}}%
             {}}%
    \fi}%
  \or % 2 = All RTL - tabular
    \let\bbl@parabefore\relax
    \AddToHook{para/before}{\bbl@parabefore}%
    \AtBeginDocument{%
      \@ifpackageloaded{colortbl}%
        {\bbl@replace\@tabular{$}{$%
           \def\bbl@insidemath{0}%
           \def\bbl@parabefore{\localerestoredirs}}%
         \bbl@sreplace\@classz
           {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
        {}}%
  \fi
```

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

```
  \AtBeginDocument{%
    \@ifpackageloaded{multicol}%
      {\toks@\expandafter{\multi@column@out}%
       \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
      {}%
    \@ifpackageloaded{paracol}%
      {\edef\pcol@output{%
        \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
      {}}%
\fi
\ifx\bbl@opt@layout\@nnil\endinput\fi  % 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#1{%  non-local changes, use always inside a group!
    \bbl@exp{%
      \mathdir\the\bodydir
      #1%             Once entered in math, set boxes to restore values
      \def\\\bbl@insidemath{0}%
      \<ifmmode>%
        \everyvbox{%
          \the\everyvbox
          \bodydir\the\bodydir
          \mathdir\the\mathdir
          \everyhbox{\the\everyhbox}%
          \everyvbox{\the\everyvbox}}%
        \everyhbox{%
          \the\everyhbox
          \bodydir\the\bodydir
          \mathdir\the\mathdir
          \everyhbox{\the\everyhbox}%
          \everyvbox{\the\everyvbox}}%
      \<fi>}}%
  \def\@hangfrom#1{%
    \setbox\@tempboxa\hbox{{#1}}%
    \hangindent\wd\@tempboxa
    \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
      \shapemode\@ne
    \fi
    \noindent\box\@tempboxa}
\fi
\IfBabelLayout{tabular}
  {\let\bbl@OL@@tabular\@tabular
   \bbl@replace\@tabular{$}{\bbl@nextfake$}%
   \let\bbl@NL@@tabular\@tabular
   \AtBeginDocument{%
     \ifx\bbl@NL@@tabular\@tabular\else
       \bbl@exp{\\\in@{\\\bbl@nextfake}{\[@tabular]}}%
       \ifin@\else
         \bbl@replace\@tabular{$}{\bbl@nextfake$}%
       \fi
       \let\bbl@NL@@tabular\@tabular
     \fi}}
   {}
\IfBabelLayout{lists}
```

```
  {\let\bbl@OL@list\list
   \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
   \let\bbl@NL@list\list
   \def\bbl@listparshape#1#2#3{%
     \parshape #1 #2 #3 %
     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
       \shapemode\tw@
     \fi}}
  {}
\IfBabelLayout{graphics}
  {\let\bbl@pictresetdir\relax
   \def\bbl@pictsetdir#1{%
     \ifcase\bbl@thetextdir
       \let\bbl@pictresetdir\relax
     \else
       \ifcase#1\bodydir TLT  % Remember this sets the inner boxes
         \or\textdir TLT
         \else\bodydir TLT \textdir TLT
       \fi
       % \(text|par)dir required in pgf:
       \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
     \fi}%
   \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
   \directlua{
     Babel.get_picture_dir = true
     Babel.picture_has_bidi = 0
     %
     function Babel.picture_dir (head)
       if not Babel.get_picture_dir then return head end
       if Babel.hlist_has_bidi(head) then
         Babel.picture_has_bidi = 1
       end
       return head
     end
     luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
       "Babel.picture_dir")
   }%
   \AtBeginDocument{%
     \def\LS@rot{%
       \setbox\@outputbox\vbox{%
         \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}}%
     \long\def\put(#1,#2)#3{%
       \@killglue
       % Try:
       \ifx\bbl@pictresetdir\relax
         \def\bbl@tempc{0}%
       \else
         \directlua{
           Babel.get_picture_dir = true
           Babel.picture_has_bidi = 0
         }%
         \setbox\z@\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@pictresetdir}%
       \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
     \fi
     \ifx\tikzpicture\@undefined\else
       \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\tw@}%
       \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
       \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
     \fi
     \ifx\tcolorbox\@undefined\else
       \def\tcb@drawing@env@begin{%
         \csname tcb@before@\tcb@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@\tcb@split@state\endcsname}%
     \fi
   }}
  {}
```

Implicitly reverses sectioning labels in bidi=basic-r, because the full stop is not in contact with L numbers any more. I think there must be a better way. Assumes `bidi=basic`, but there are some additional readjustments for `bidi=default`.

```
\IfBabelLayout{counters*}%
  {\bbl@add\bbl@opt@layout{.counters.}%
   \directlua{
     luatexbase.add_to_callback("process_output_buffer",
       Babel.discard_sublr , "Babel.discard_sublr") }%
  }{}
\IfBabelLayout{counters}%
  {\let\bbl@OL@@textsuperscript\@textsuperscript
   \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
   \let\bbl@latinarabic=\@arabic
   \let\bbl@OL@@arabic\@arabic
   \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
   \@ifpackagewith{babel}{bidi=default}%
     {\let\bbl@asciiroman=\@roman
      \let\bbl@OL@@roman\@roman
      \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
      \let\bbl@asciiRoman=\@Roman
      \let\bbl@OL@@roman\@Roman
      \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
      \let\bbl@OL@labelenumii\labelenumii
      \def\labelenumii{)\theenumii(}%
      \let\bbl@OL@p@enumiii\p@enumiii
      \def\p@enumiii{\p@enumii)\theenumii(}}{}}{}
⟨⟨Footnote changes⟩⟩
\IfBabelLayout{footnotes}%
  {\let\bbl@OL@footnote\footnote
   \BabelFootnote\footnote\languagename{}{}%
   \BabelFootnote\localfootnote\languagename{}{}%
   \BabelFootnote\mainfootnote{}{}{}}
  {}
```

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}%
  {\bbl@ncarg\let\bbl@OL@underline{underline }%
   \bbl@carg\bbl@sreplace{underline }%
     {$\@@underline}{\bgroup\bbl@nextfake$\@@underline}%
   \bbl@carg\bbl@sreplace{underline }%
     {\m@th$}{\m@th$\egroup}%
   \let\bbl@OL@LaTeXe\LaTeXe
   \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
     \if b\expandafter\@car\f@series\@nil\boldmath\fi
     \babelsublr{%
       \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
  {}
⟨/luatex⟩
```

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 discretionaries, which `pattern` is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck).
`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.

```
⟨*transforms⟩
Babel.linebreaking.replacements = {}
Babel.linebreaking.replacements[0] = {}  -- pre
Babel.linebreaking.replacements[1] = {}  -- post

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

Babel.fetch_subtext = {}

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

-- Merging both functions doesn't seen feasible, because there are too
-- many differences.
Babel.fetch_subtext[0] = function(head)
```

```
  local word_string = ''
  local word_nodes = {}
  local lang
  local item = head
  local inmath = false

  while item do

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

    if inmath then
      -- pass

    elseif item.id == 29 then
      local locale = node.get_attribute(item, Babel.attr_locale)

      if lang == locale or lang == nil then
        lang = lang or locale
        if Babel.ignore_pre_char(item) then
          word_string = word_string .. Babel.us_char
        else
          word_string = word_string .. unicode.utf8.char(item.char)
        end
        word_nodes[#word_nodes+1] = item
      else
        break
      end

    elseif item.id == 12 and item.subtype == 13 then
      word_string = word_string .. ' '
      word_nodes[#word_nodes+1] = item

    -- Ignore leading unrecognized nodes, too.
    elseif word_string ~= '' then
      word_string = word_string .. Babel.us_char
      word_nodes[#word_nodes+1] = item  -- Will be ignored
    end

    item = item.next
  end

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

Babel.fetch_subtext[1] = function(head)
  local word_string = ''
  local word_nodes = {}
  local lang
  local item = head
  local inmath = false

  while item do

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

    if inmath then
      -- pass

    elseif item.id == 29 then
      if item.lang == lang or lang == nil then
        if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
          lang = lang or item.lang
          word_string = word_string .. unicode.utf8.char(item.char)
          word_nodes[#word_nodes+1] = item
        end
      else
        break
      end

    elseif item.id == 7 and item.subtype == 2 then
      word_string = word_string .. '='
      word_nodes[#word_nodes+1] = item

    elseif item.id == 7 and item.subtype == 3 then
      word_string = word_string .. '|'
      word_nodes[#word_nodes+1] = item

    -- (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
  end

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

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

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

Babel.us_char = string.char(31)

function Babel.hyphenate_replace(head, mode)
  local u = unicode.utf8
  local lbkr = Babel.linebreaking.replacements[mode]

  local word_head = head

  while true do  -- for each subtext block

```

```
    local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)

    if Babel.debug then
      print()
      print((mode == 0) and '@@@@<' or '@@@@>', w)
    end

    if nw == nil and w == '' then break end

    if not lang then goto next end
    if not lbkr[lang] then goto next end

    -- For each saved (pre|post)hyphenation. TODO. Reconsider how
    -- loops are nested.
    for k=1, #lbkr[lang] do
      local p = lbkr[lang][k].pattern
      local r = lbkr[lang][k].replace
      local attr = lbkr[lang][k].attr or -1

      if Babel.debug then
        print('*****', p, mode)
      end

      -- This variable is set in some cases below to the first *byte*
      -- after the match, either as found by u.match (faster) or the
      -- computed position based on sc if w has changed.
      local last_match = 0
      local step = 0

      -- For every match.
      while true do
        if Babel.debug then
          print('=====')
        end
        local new  -- used when inserting and removing nodes
        local 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

          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

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

          elseif crep and crep.kashida then -- Experimental
            node.set_attribute(item,
               Babel.attr_kashida,
               crep.kashida)
            last_match = utf8.offset(w, sc+1+step)
            goto next

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

          elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
            d = node.new(7, 3)   -- (disc, regular)
            d.pre     = Babel.str_to_nodes(crep.pre, matches, item_base)
            d.post    = Babel.str_to_nodes(crep.post, matches, item_base)
            d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
            d.attr = item_base.attr
            if crep.pre == nil then  -- TeXbook p96
              d.penalty = crep.penalty or tex.hyphenpenalty
            else
              d.penalty = crep.penalty or tex.exhyphenpenalty
            end
            placeholder = '|'
            head, new = node.insert_before(head, item, d)

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

          elseif crep and crep.penalty then
            d = node.new(14, 0)   -- (penalty, userpenalty)
            d.attr = item_base.attr
            d.penalty = crep.penalty
            head, new = node.insert_before(head, item, d)

          elseif crep and crep.space then
            -- 655360 = 10 pt = 10 * 65536 sp
            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

          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
  end

  -- This table stores capture maps, numbered consecutively
  Babel.capture_maps = {}

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

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

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

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

  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
        res = res .. ' '
      elseif n.id == 29 then
        res = res .. unicode.utf8.char(n.char)
      end
    end
    tex.print(res)
  end
  ⟨/transforms⟩
```

10.12 Lua: Auto bidi with `basic` and `basic-r`

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

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

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

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

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, *what* they do and *why*, and not only *how*), but I think (or I hope) I've managed to understand them. In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually *two* R modes (set

explicitly in Unicode with RLM and ALM). In babel the dir is set by a higher protocol based on the language/script, which in turn sets the correct dir (<l>, <r> or <al>).
From UAX#9: “Where available, markup should be used instead of the explicit formatting characters”. So, this simple version just ignores formatting characters. Actually, most of that annex is devoted to how to handle them.
BD14-BD16 are not implemented. Unicode (and the W3C) are making a great effort to deal with some special problematic cases in “streamed” plain text. I don’t think this is the way to go – particular issues should be fixed by a high level interface taking into account the needs of the document. And here is where luatex excels, because everything related to bidi writing is under our control.

```
⟨*basic-r⟩
Babel = Babel or {}

Babel.bidi_enabled = true

require('babel-data-bidi.lua')

local characters = Babel.characters
local ranges = Babel.ranges

local DIR = node.id("dir")

local function dir_mark(head, from, to, outer)
  dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
  local d = node.new(DIR)
  d.dir = '+' .. dir
  node.insert_before(head, from, d)
  d = node.new(DIR)
  d.dir = '-' .. dir
  node.insert_after(head, to, d)
end

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

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

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

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

  local last_lr

  local type_n = ''

  for item in node.traverse(head) do

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

      local itemchar
      if item.id == 7 and item.subtype == 2 then
        itemchar = item.replace.char
      else
        itemchar = item.char
      end
      local chardata = characters[itemchar]
      dir = chardata and chardata.d or nil
      if not dir then
        for nn, et in ipairs(ranges) do
          if itemchar < et[1] then
            break
          elseif itemchar <= et[2] then
            dir = et[3]
            break
          end
        end
      end
      dir = dir or 'l'
      if inmath then dir = ('TRT' == tex.mathdir) and 'r' or 'l' end
```

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

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

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

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

```
      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, a whatsit, but this is what I would expect (with luacolor you may colorize some digits). Anyway, this behavior could be changed with a switch in the future. Note in the first branch only <an> is relevant if <al>.

```
    if dir == 'en' or dir == 'an' or dir == 'et' then
      if dir ~= 'et' then
        type_n = dir
      end
      first_n = first_n or item
      last_n = last_es or item
      last_es = nil
    elseif dir == 'es' and last_n then -- W3+W6
      last_es = item
    elseif dir == 'cs' then            -- it's right - do nothing
    elseif first_n then -- & if dir = any but en, et, 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, whatsits, 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>, resptly, 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. Numbers in R mode are processed. It should not be done, but it doesn't hurt.

```
    if dir and not last_lr and dir ~= 'l' and outer == 'r' then
      item.char = characters[item.char] and
                  characters[item.char].m or item.char
    elseif (dir or new_dir) and last_lr ~= item then
      local mir = outer .. strong_lr .. (dir or outer)
      if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
        for ch in node.traverse(node.next(last_lr)) do
          if ch == item then break end
          if ch.id == node.id'glyph' and characters[ch.char] then
            ch.char = characters[ch.char].m or ch.char
          end
        end
      end
    end
```

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

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

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

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

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

```
  return node.prev(head) or head
end
⟨/basic-r⟩
```

And here the Lua code for `bidi=basic`:

```
⟨*basic⟩
Babel = Babel or {}

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

Babel.fontmap = Babel.fontmap or {}
Babel.fontmap[0] = {}      -- l
Babel.fontmap[1] = {}      -- r
Babel.fontmap[2] = {}      -- al/an

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

Babel.bidi_enabled = true
Babel.mirroring_enabled = true

require('babel-data-bidi.lua')

local characters = Babel.characters
local ranges = Babel.ranges

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

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

local function insert_numeric(head, state)
  local new
  local new_state = state
  if state.san and state.ean and state.san ~= state.ean then
    local d = node.new(DIR)
    d.dir = '+TLT'
    _, new = node.insert_before(head, state.san, d)
    if state.san == state.sim then state.sim = new end
    local d = node.new(DIR)
    d.dir = '-TLT'
    _, new = node.insert_after(head, state.ean, d)
    if state.ean == state.eim then state.eim = new end
  end
  new_state.san, new_state.ean = nil, nil
  return head, new_state
end

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

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

function Babel.bidi(head, ispar, hdir)
  local d   -- d is used mainly for computations in a loop
  local prev_d = ''
  local new_d = false

  local nodes = {}
  local outer_first = nil
  local inmath = false

  local glue_d = nil
  local glue_i = nil

  local has_en = false
  local first_et = nil

  local has_hyperlink = false

  local ATDIR = Babel.attr_dir
  local attr_d

  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 = 'r' end

  local fontmap = Babel.fontmap

  for item in node.traverse(head) do

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

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

      if node.get_attribute(item, ATDIR) == 128 then goto nextnode end

      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})
      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

  node.set_attribute(item, ATDIR, 128)
  outer_first = nil
```

```

  ::nextnode::

end -- for each node

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

outer = save_outer
last = outer

local first_on = nil

for q = 1, #nodes do
  local item

  local outer_first = nodes[q][3]
  outer = outer_first or outer
  last = outer_first or last

  local d = nodes[q][2]
  if d == 'an' or d == 'en' then d = 'r' end
  if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6

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

  if d == 'r' or d == 'l' then last = d end
end

--------------  IMPLICIT, REORDER ----------------

outer = save_outer
last = outer

local state = {}
state.has_r = false

for q = 1, #nodes do

  local item = nodes[q][1]

  outer = nodes[q][3] or outer

  local d = nodes[q][2]

  if d == 'nsm' then d = last end             -- W1
  if d == 'en' then d = 'an' end
  local isdir = (d == 'r' or d == 'l')

  if outer == 'l' and d == 'an' then
    state.san = state.san or item
    state.ean = item
  elseif state.san then
    head, state = insert_numeric(head, state)
  end

  if outer == 'l' then
    if d == 'an' or d == 'r' then     -- im -> implicit
      if d == 'r' then state.has_r = true end
      state.sim = state.sim or item
      state.eim = item
    elseif d == 'l' and state.sim and state.has_r then
      head, state = insert_implicit(head, state, outer)
    elseif d == 'l' then
      state.sim, state.eim, state.has_r = nil, nil, false
    end
  else
    if d == 'an' or d == 'l' then
      if nodes[q][3] then -- nil except after an explicit dir
        state.sim = item  -- so we move sim 'inside' the group
      else
        state.sim = state.sim or item
      end
      state.eim = item
    elseif d == 'r' and state.sim then
      head, state = insert_implicit(head, state, outer)
    elseif d == 'r' then
      state.sim, state.eim = nil, nil
    end
  end

  if isdir then
    last = d           -- Don't search back - best save now
  elseif 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

  return head
end
-- Make sure anything is marked as 'bidi done' (including nodes inserted
-- after the babel algorithm).
function Babel.unset_atdir(head)
  local ATDIR = Babel.attr_dir
  for item in node.traverse(head) do
    node.set_attribute(item, ATDIR, 128)
  end
  return head
end
⟨/basic⟩
```

11 Data for CJK

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

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

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

12 The ‘nil’ language

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

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

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

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

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

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

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

\captionnil
\datenil

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

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

```
\def\bbl@inidata@nil{%
  \bbl@elt{identification}{tag.ini}{und}%
  \bbl@elt{identification}{load.level}{0}%
  \bbl@elt{identification}{charset}{utf8}%
  \bbl@elt{identification}{version}{1.0}%
  \bbl@elt{identification}{date}{2022-05-16}%
  \bbl@elt{identification}{name.local}{nil}%
  \bbl@elt{identification}{name.english}{nil}%
  \bbl@elt{identification}{name.babel}{nil}%
  \bbl@elt{identification}{tag.bcp47}{und}%
  \bbl@elt{identification}{language.tag.bcp47}{und}%
  \bbl@elt{identification}{tag.opentype}{dflt}%
  \bbl@elt{identification}{script.name}{Latin}%
  \bbl@elt{identification}{script.tag.bcp47}{Latn}%
  \bbl@elt{identification}{script.tag.opentype}{DFLT}%
  \bbl@elt{identification}{level}{1}%
  \bbl@elt{identification}{encodings}{}%
  \bbl@elt{identification}{derivate}{no}}
\@namedef{bbl@tbcp@nil}{und}
\@namedef{bbl@lbcp@nil}{und}
\@namedef{bbl@casing@nil}{und} % TODO
\@namedef{bbl@lotf@nil}{dflt}
\@namedef{bbl@elname@nil}{nil}
\@namedef{bbl@lname@nil}{nil}
\@namedef{bbl@esname@nil}{Latin}
\@namedef{bbl@sname@nil}{Latin}
\@namedef{bbl@sbcp@nil}{Latn}
\@namedef{bbl@sotf@nil}{latn}
```

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

```
\ldf@finish{nil}
⟨/nil⟩
```

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.

```
⟨⟨*Compute Julian day⟩⟩ ≡
\def\bbl@fpmod#1#2{(#1-#2*floor(#1/#2))}
\def\bbl@cs@gregleap#1{%
  (\bbl@fpmod{#1}{4} == 0) &&
    (!((\bbl@fpmod{#1}{100} == 0) && (\bbl@fpmod{#1}{400} != 0)))}
\def\bbl@cs@jd#1#2#3{% year, month, day
  \fp_eval:n{ 1721424.5   + (365 * (#1 - 1)) +
    floor((#1 - 1) / 4)   + (-floor((#1 - 1) / 100)) +
    floor((#1 - 1) / 400) + floor((((367 * #2) - 362) / 12) +
    ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3) }}
⟨⟨/Compute Julian day⟩⟩
```

13.1 Islamic

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

```
⟨*ca-islamic⟩
\ExplSyntaxOn
⟨⟨Compute Julian day⟩⟩
% == islamic (default)
% Not yet implemented
\def\bbl@ca@islamic#1-#2-#3\@@#4#5#6{}
```

The Civil calendar.

```
\def\bbl@cs@isltojd#1#2#3{ % year, month, day
  ((#3 + ceil(29.5 * (#2 - 1)) +
  (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
  1948439.5) - 1) }
\@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{}}
\@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
\@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
\def\bbl@ca@islamicvl@x#1#2-#3-#4\@@#5#6#7{%
  \edef\bbl@tempa{%
    \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
  \edef#5{%
    \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
  \edef#6{\fp_eval:n{
    min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
  \edef#7{\fp_eval:n{ \bbl@tempa - \bbl@cs@isltojd{#5}{#6}{1} + 1} }}
```

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,%
  61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
  62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
  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}
\@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
\@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
\@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
\def\bbl@ca@islamcuqr@x#1#2-#3-#4\@@#5#6#7{%
  \ifnum#2>2014 \ifnum#2<2038
    \bbl@afterfi\expandafter\@gobble
  \fi\fi
    {\bbl@error{year-out-range}{2014-2038}{}{}}%
  \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
    \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
  \count@\@ne
  \bbl@foreach\bbl@cs@umalqura@data{%
    \advance\count@\@ne
    \ifnum##1>\bbl@tempd\else
      \edef\bbl@tempe{\the\count@}%
      \edef\bbl@tempb{##1}%
    \fi}%
  \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
  \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1 ) / 12) }}% annus
  \edef#5{\fp_eval:n{ \bbl@tempa + 1  }}%
  \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
  \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}}
\ExplSyntaxOff
\bbl@add\bbl@precalendar{%
  \bbl@replace\bbl@ld@calendar{-civil}{}%
  \bbl@replace\bbl@ld@calendar{-umalqura}{}%
  \bbl@replace\bbl@ld@calendar{+}{}%
  \bbl@replace\bbl@ld@calendar{-}{}}
⟨/ca-islamic⟩
```

13.2 Hebrew

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

```
⟨*ca-hebrew⟩
\newcount\bbl@cntcommon
\def\bbl@remainder#1#2#3{%
  #3=#1\relax
  \divide #3 by #2\relax
  \multiply #3 by -#2\relax
  \advance #3 by #1\relax}%
\newif\ifbbl@divisible
\def\bbl@checkifdivisible#1#2{%
```

```
 {\countdef\tmp=0
  \bbl@remainder{#1}{#2}{\tmp}%
  \ifnum \tmp=0
      \global\bbl@divisibletrue
  \else
      \global\bbl@divisiblefalse
  \fi}}
\newif\ifbbl@gregleap
\def\bbl@ifgregleap#1{%
  \bbl@checkifdivisible{#1}{4}%
  \ifbbl@divisible
      \bbl@checkifdivisible{#1}{100}%
      \ifbbl@divisible
          \bbl@checkifdivisible{#1}{400}%
          \ifbbl@divisible
              \bbl@gregleaptrue
          \else
              \bbl@gregleapfalse
          \fi
      \else
          \bbl@gregleaptrue
      \fi
  \else
      \bbl@gregleapfalse
  \fi
  \ifbbl@gregleap}
\def\bbl@gregdayspriormonths#1#2#3{%
    {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
          181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
     \bbl@ifgregleap{#2}%
         \ifnum #1 > 2
             \advance #3 by 1
         \fi
     \fi
     \global\bbl@cntcommon=#3}%
    #3=\bbl@cntcommon}
\def\bbl@gregdaysprioryears#1#2{%
  {\countdef\tmpc=4
   \countdef\tmpb=2
   \tmpb=#1\relax
   \advance \tmpb by -1
   \tmpc=\tmpb
   \multiply \tmpc by 365
   #2=\tmpc
   \tmpc=\tmpb
   \divide \tmpc by 4
   \advance #2 by \tmpc
   \tmpc=\tmpb
   \divide \tmpc by 100
   \advance #2 by -\tmpc
   \tmpc=\tmpb
   \divide \tmpc by 400
   \advance #2 by \tmpc
   \global\bbl@cntcommon=#2\relax}%
  #2=\bbl@cntcommon}
\def\bbl@absfromgreg#1#2#3#4{%
  {\countdef\tmpd=0
   #4=#1\relax
   \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
   \advance #4 by \tmpd
   \bbl@gregdaysprioryears{#3}{\tmpd}%
   \advance #4 by \tmpd
   \global\bbl@cntcommon=#4\relax}%
  #4=\bbl@cntcommon}
\newif\ifbbl@hebrleap
\def\bbl@checkleaphebryear#1{%
  {\countdef\tmpa=0
   \countdef\tmpb=1
   \tmpa=#1\relax
   \multiply \tmpa by 7
   \advance \tmpa by 1
   \bbl@remainder{\tmpa}{19}{\tmpb}%
   \ifnum \tmpb < 7
       \global\bbl@hebrleaptrue
   \else
       \global\bbl@hebrleapfalse
   \fi}}
\def\bbl@hebrelapsedmonths#1#2{%
  {\countdef\tmpa=0
   \countdef\tmpb=1
   \countdef\tmpc=2
   \tmpa=#1\relax
   \advance \tmpa by -1
   #2=\tmpa
   \divide #2 by 19
   \multiply #2 by 235
   \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
   \tmpc=\tmpb
   \multiply \tmpb by 12
   \advance #2 by \tmpb
   \multiply \tmpc by 7
   \advance \tmpc by 1
   \divide \tmpc by 19
   \advance #2 by \tmpc
   \global\bbl@cntcommon=#2}%
  #2=\bbl@cntcommon}
\def\bbl@hebrelapseddays#1#2{%
  {\countdef\tmpa=0
   \countdef\tmpb=1
   \countdef\tmpc=2
   \bbl@hebrelapsedmonths{#1}{#2}%
   \tmpa=#2\relax
   \multiply \tmpa by 13753
   \advance \tmpa by 5604
   \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
   \divide \tmpa by 25920
   \multiply #2 by 29
   \advance #2 by 1
   \advance #2 by \tmpa
   \bbl@remainder{#2}{7}{\tmpa}%
   \ifnum \tmpc < 19440
       \ifnum \tmpc < 9924
       \else
           \ifnum \tmpa=2
               \bbl@checkleaphebryear{#1}% of a common year
               \ifbbl@hebrleap
               \else
                   \advance #2 by 1
               \fi
           \fi
       \fi
       \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
  \bbl@remainder{#2}{7}{\tmpa}%
  \ifnum \tmpa=0
    \advance #2 by 1
  \else
    \ifnum \tmpa=3
      \advance #2 by 1
    \else
      \ifnum \tmpa=5
          \advance #2 by 1
      \fi
    \fi
  \fi
  \global\bbl@cntcommon=#2\relax}%
 #2=\bbl@cntcommon}
\def\bbl@daysinhebryear#1#2{%
 {\countdef\tmpe=12
  \bbl@hebrelapseddays{#1}{\tmpe}%
  \advance #1 by 1
  \bbl@hebrelapseddays{#1}{#2}%
  \advance #2 by -\tmpe
  \global\bbl@cntcommon=#2}%
 #2=\bbl@cntcommon}
\def\bbl@hebrdayspriormonths#1#2#3{%
 {\countdef\tmpf= 14
  #3=\ifcase #1\relax
         0 \or
         0 \or
        30 \or
        59 \or
        89 \or
       118 \or
       148 \or
       148 \or
       177 \or
       207 \or
       236 \or
       266 \or
       295 \or
       325 \or
       400
  \fi
  \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}%
      \ifnum \tmpx < #4\relax
          \advance #5 by 1
          \tmpy=\tmpx
  \repeat
  \global\advance #5 by -1
  \global\advance #4 by -\tmpy}}
\newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
\newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
\def\bbl@ca@hebrew#1-#2-#3\@@#4#5#6{%
 \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
 \bbl@hebrfromgreg
   {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
   {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
 \edef#4{\the\bbl@hebryear}%
 \edef#5{\the\bbl@hebrmonth}%
 \edef#6{\the\bbl@hebrday}}
⟨/ca-hebrew⟩
```

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).

```
⟨*ca-persian⟩
\ExplSyntaxOn
⟨⟨Compute Julian day⟩⟩
\def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
  2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
\def\bbl@ca@persian#1-#2-#3\@@#4#5#6{%
  \edef\bbl@tempa{#1}%  20XX-03-\bbl@tempe = 1 farvardin:
  \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
    \bbl@afterfi\expandafter\@gobble
  \fi\fi
    {\bbl@error{year-out-range}{2013-2050}{}{}}%
  \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
  \ifin@\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
  \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
  \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
  \ifnum\bbl@tempc<\bbl@tempb
    \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
    \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
    \ifin@\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
    \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
  \fi
  \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
  \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
  \edef#5{\fp_eval:n{% set Jalali month
    (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
  \edef#6{\fp_eval:n{% set Jalali day
    (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6)))}}}
\ExplSyntaxOff
⟨/ca-persian⟩
```

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.

```
⟨*ca-coptic⟩
\ExplSyntaxOn
⟨⟨Compute Julian day⟩⟩
\def\bbl@ca@coptic#1-#2-#3\@@#4#5#6{%
  \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
  \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
  \edef#4{\fp_eval:n{%
    floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
  \edef\bbl@tempc{\fp_eval:n{%
     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
  \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
  \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}
\ExplSyntaxOff
⟨/ca-coptic⟩
⟨*ca-ethiopic⟩
\ExplSyntaxOn
⟨⟨Compute Julian day⟩⟩
\def\bbl@ca@ethiopic#1-#2-#3\@@#4#5#6{%
  \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
  \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
  \edef#4{\fp_eval:n{%
    floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
  \edef\bbl@tempc{\fp_eval:n{%
     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
  \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
  \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}
\ExplSyntaxOff
```

```
⟨/ca-ethiopic⟩
```

13.5 Buddhist

That’s very simple.

```
⟨*ca-buddhist⟩
\def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
  \edef#4{\number\numexpr#1+543\relax}%
  \edef#5{#2}%
  \edef#6{#3}}
⟨/ca-buddhist⟩
%
% \subsection{Chinese}
%
% Brute force, with the Julian day of first day of each month. The
% table has been computed with the help of \textsf{python-lunardate} by
% Ricky Yeung, GPLv2 (but the code itself has not been used). The range
% is 2015-2044.
%
%    \begin{macrocode}
⟨*ca-chinese⟩
\ExplSyntaxOn
⟨⟨Compute Julian day⟩⟩
\def\bbl@ca@chinese#1-#2-#3\@@#4#5#6{%
  \edef\bbl@tempd{\fp_eval:n{%
    \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
  \count@\z@
  \@tempcnta=2015
  \bbl@foreach\bbl@cs@chinese@data{%
    \ifnum##1>\bbl@tempd\else
      \advance\count@\@ne
      \ifnum\count@>12
        \count@\@ne
        \advance\@tempcnta\@ne\fi
      \bbl@xin@{,##1,}{,\bbl@cs@chinese@leap,}%
      \ifin@
        \advance\count@\m@ne
        \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
      \else
        \edef\bbl@tempe{\the\count@}%
      \fi
      \edef\bbl@tempb{##1}%
    \fi}%
  \edef#4{\the\@tempcnta}%
  \edef#5{\bbl@tempe}%
  \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
\def\bbl@cs@chinese@leap{%
  885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
\def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
  354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
  768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
  1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
  1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
  1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
  2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
  2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
  2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
  3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
  3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
  3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
  4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
  4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
  5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
10896,10926,10956,10986,11015,11045,11074,11103}
\ExplSyntaxOff
⟨/ca-chinese⟩
```

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

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to achieve the desired effect, based on the `babel` package. If you load each of them with iniTEX, you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`.
As these files are going to be read as the first thing iniTEX sees, we need to set some category codes just to be able to change the definition of `\input`.

```
⟨*bplain | blplain⟩
\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).

```
\openin 0 hyphen.cfg
\ifeof0
\else
  \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
⟨/bplain | blplain⟩
```

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
⟨blplain⟩\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.

```
⟨bplain⟩\def\fmtname{babel-plain}
⟨blplain⟩\def\fmtname{babel-lplain}
```

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

14.2 Emulating some LaTeX features

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

```
⟨⟨*Emulate LaTeX⟩⟩ ≡
\def\@empty{}
\def\loadlocalcfg#1{%
  \openin0#1.cfg
  \ifeof0
    \closein0
  \else
    \closein0
    {\immediate\write16{*************************************}%
     \immediate\write16{* Local config file #1.cfg used}%
     \immediate\write16{*}%
     }
    \input #1.cfg\relax
  \fi
  \@endofldf}
```

14.3 General tools

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

```
\long\def\@firstofone#1{#1}
\long\def\@firstoftwo#1#2{#1}
\long\def\@secondoftwo#1#2{#2}
\def\@nnil{\@nil}
\def\@gobbletwo#1#2{}
\def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
\def\@star@or@long#1{%
  \@ifstar
  {\let\l@ngrel@x\relax#1}%
  {\let\l@ngrel@x\long#1}}
\let\l@ngrel@x\relax
\def\@car#1#2\@nil{#1}
\def\@cdr#1#2\@nil{#2}
\let\@typeset@protect\relax
\let\protected@edef\edef
\long\def\@gobble#1{}
\edef\@backslashchar{\expandafter\@gobble\string\\}
\def\strip@prefix#1>{}
\def\g@addto@macro#1#2{{%
    \toks@\expandafter{#1#2}%
    \xdef#1{\the\toks@}}}
\def\@namedef#1{\expandafter\def\csname #1\endcsname}
\def\@nameuse#1{\csname #1\endcsname}
```

```
\def\@ifundefined#1{%
  \expandafter\ifx\csname#1\endcsname\relax
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi}
\def\@expandtwoargs#1#2#3{%
  \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
\def\zap@space#1 #2{%
  #1%
  \ifx#2\@empty\else\expandafter\zap@space\fi
  #2}
\let\bbl@trace\@gobble
\def\bbl@error#1{% Implicit #2#3#4
  \begingroup
    \catcode`\\=0   \catcode`\==12 \catcode`\`=12
    \catcode`\^^M=5 \catcode`\%=14
    \input errbabel.def
  \endgroup
  \bbl@error{#1}}
\def\bbl@warning#1{%
  \begingroup
    \newlinechar=`\^^J
    \def\\{^^J(babel) }%
    \message{\\#1}%
  \endgroup}
\let\bbl@infowarn\bbl@warning
\def\bbl@info#1{%
  \begingroup
    \newlinechar=`\^^J
    \def\\{^^J}%
    \wlog{#1}%
  \endgroup}
```

LaTeX 2_ε has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```
\ifx\@preamblecmds\@undefined
  \def\@preamblecmds{}
\fi
\def\@onlypreamble#1{%
  \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
    \@preamblecmds\do#1}}
\@onlypreamble\@onlypreamble
```

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 `\ifx`. The same trick is applied below.

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

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\expandafter\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}%
    \advance\@tempcntb \@ne}%
  \let\@hash@##%
  \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
\def\providecommand{\@star@or@long\provide@command}
\def\provide@command#1{%
  \begingroup
    \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
  \endgroup
  \expandafter\@ifundefined\@gtempa
    {\def\reserved@a{\new@command#1}}%
    {\let\reserved@a\relax
     \def\reserved@a{\new@command\reserved@a}}%
   \reserved@a}%
```

```
\def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
\def\declare@robustcommand#1{%
   \edef\reserved@a{\string#1}%
   \def\reserved@b{#1}%
   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
   \edef#1{%
      \ifx\reserved@a\reserved@b
         \noexpand\x@protect
         \noexpand#1%
      \fi
      \noexpand\protect
      \expandafter\noexpand\csname
         \expandafter\@gobble\string#1 \endcsname
   }%
   \expandafter\new@command\csname
      \expandafter\@gobble\string#1 \endcsname
}
\def\x@protect#1{%
   \ifx\protect\@typeset@protect\else
      \@x@protect#1%
   \fi
}
\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
```

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

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

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

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

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

```
\ifx\@tempcnta\@undefined
  \csname newcount\endcsname\@tempcnta\relax
\fi
\ifx\@tempcntb\@undefined
  \csname newcount\endcsname\@tempcntb\relax
\fi
```

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

```
\ifx\bye\@undefined
  \advance\count10 by -2\relax
\fi
\ifx\@ifnextchar\@undefined
  \def\@ifnextchar#1#2#3{%
    \let\reserved@d=#1%
    \def\reserved@a{#2}\def\reserved@b{#3}%
    \futurelet\@let@token\@ifnch}
  \def\@ifnch{%
    \ifx\@let@token\@sptoken
      \let\reserved@c\@xifnch
    \else
      \ifx\@let@token\reserved@d
        \let\reserved@c\reserved@a
      \else
        \let\reserved@c\reserved@b
      \fi
    \fi
    \reserved@c}
  \def\:{\let\@sptoken= } \:  % this makes \@sptoken a space token
  \def\:{\@xifnch} \expandafter\def\: {\futurelet\@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}}
```

14.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain TEX environment.

```
\def\DeclareTextCommand{%
   \@dec@text@cmd\providecommand
}
\def\ProvideTextCommand{%
   \@dec@text@cmd\providecommand
}
\def\DeclareTextSymbol#1#2#3{%
   \@dec@text@cmd\chardef#1{#2}#3\relax
}
\def\@dec@text@cmd#1#2#3{%
   \expandafter\def\expandafter#2%
      \expandafter{%
         \csname#3-cmd\expandafter\endcsname
         \expandafter#2%
         \csname#3\string#2\endcsname
      }%
%   \let\@ifdefinable\@rc@ifdefinable
   \expandafter#1\csname#3\string#2\endcsname
}
\def\@current@cmd#1{%
  \ifx\protect\@typeset@protect\else
      \noexpand#1\expandafter\@gobble
  \fi
}
\def\@changed@cmd#1#2{%
   \ifx\protect\@typeset@protect
      \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
         \expandafter\ifx\csname ?\string#1\endcsname\relax
            \expandafter\def\csname ?\string#1\endcsname{%
               \@changed@x@err{#1}%
            }%
         \fi
         \global\expandafter\let
           \csname\cf@encoding \string#1\expandafter\endcsname
           \csname ?\string#1\endcsname
      \fi
      \csname\cf@encoding\string#1%
        \expandafter\endcsname
   \else
```

```
      \noexpand#1%
   \fi
}
\def\@changed@x@err#1{%
    \errhelp{Your command will be ignored, type <return> to proceed}%
    \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
\def\DeclareTextCommandDefault#1{%
   \DeclareTextCommand#1?%
}
\def\ProvideTextCommandDefault#1{%
   \ProvideTextCommand#1?%
}
\expandafter\let\csname OT1-cmd\endcsname\@current@cmd
\expandafter\let\csname?-cmd\endcsname\@changed@cmd
\def\DeclareTextAccent#1#2#3{%
  \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
}
\def\DeclareTextCompositeCommand#1#2#3#4{%
   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
   \edef\reserved@b{\string##1}%
   \edef\reserved@c{%
     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
   \ifx\reserved@b\reserved@c
      \expandafter\expandafter\expandafter\ifx
         \expandafter\@car\reserved@a\relax\relax\@nil
         \@text@composite
      \else
         \edef\reserved@b##1{%
            \def\expandafter\noexpand
               \csname#2\string#1\endcsname####1{%
               \noexpand\@text@composite
                  \expandafter\noexpand\csname#2\string#1\endcsname
                  ####1\noexpand\@empty\noexpand\@text@composite
                  {##1}%
            }%
         }%
         \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
      \fi
      \expandafter\def\csname\expandafter\string\csname
         #2\endcsname\string#1-\string#3\endcsname{#4}
   \else
     \errhelp{Your command will be ignored, type <return> to proceed}%
     \errmessage{\string\DeclareTextCompositeCommand\space used on
         inappropriate command \protect#1}
   \fi
}
\def\@text@composite#1#2#3\@text@composite{%
   \expandafter\@text@composite@x
      \csname\string#1-\string#2\endcsname
}
\def\@text@composite@x#1#2{%
   \ifx#1\relax
      #2%
   \else
      #1%
   \fi
}
%
\def\@strip@args#1:#2-#3\@strip@args{#2}
\def\DeclareTextComposite#1#2#3#4{%
   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
   \bgroup
      \lccode`\@=#4%
      \lowercase{%
    \egroup
      \reserved@a @%
    }%
}
%
\def\UseTextSymbol#1#2{#2}
\def\UseTextAccent#1#2#3{}
\def\@use@text@encoding#1{}
\def\DeclareTextSymbolDefault#1#2{%
    \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
}
\def\DeclareTextAccentDefault#1#2{%
    \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
}
\def\cf@encoding{OT1}
```

Currently we only use the LaTeX 2$_\varepsilon$ method for accents for those that are known to be made active in *some* language definition file.

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

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

```
\DeclareTextSymbol{\textquotedblleft}{OT1}{92}
\DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
\DeclareTextSymbol{\textquoteleft}{OT1}{`\`}
\DeclareTextSymbol{\textquoteright}{OT1}{`\'}
\DeclareTextSymbol{\i}{OT1}{16}
\DeclareTextSymbol{\ss}{OT1}{25}
```

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

```
\ifx\scriptsize\@undefined
  \let\scriptsize\sevenrm
\fi
```

And a few more "dummy" definitions.

```
\def\languagename{english}%
\let\bbl@opt@shorthands\@nnil
\def\bbl@ifshorthand#1#2#3{#2}%
\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
\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 β-testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.
There are also many contributors for specific languages, which are mentioned in the respective files. Without them, babel just wouldn't exist.

References

[1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.

[2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national LaTeX styles*, *TUGboat* 10 (1989) #3, p. 401–406.

[3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.

[4] Donald E. Knuth, *The TeXbook*, Addison-Wesley, 1986.

[5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.

[6] Leslie Lamport, *LaTeX, A document preparation System*, Addison-Wesley, 1986.

[7] Leslie Lamport, in: TeXhax Digest, Volume 89, #13, 17 February 1989.

[8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.

[9] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018

[10] Hubert Partl, *German TeX*, *TUGboat* 9 (1988) #1, p. 70–72.

[11] Joachim Schrod, *International LaTeX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.

[12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using LaTeX*, Springer, 2002, p. 301–373.

[13] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).