Babel

Localization and internationalization

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

### I  User guide

1. The user interface
   1.1 Monolingual documents ........................................ 4
   1.2 Multilingual documents ........................................ 6
   1.3 Mostly monolingual documents ............................... 7
   1.4 Modifiers .................................................. 8
   1.5 Troubleshooting ............................................ 8
   1.6 Plain ...................................................... 9
   1.7 Basic language selectors .................................. 9
   1.8 Auxiliary language selectors ............................. 10
   1.9 More on selection ......................................... 10
   1.10 Shorthands ................................................ 12
   1.11 Package options ........................................... 15
   1.12 The base option ........................................... 17
   1.13 ini files .................................................. 17
   1.14 Selecting fonts ............................................ 25
   1.15 Modifying a language .................................... 27
   1.16 Creating a language ..................................... 28
   1.17 Digits and counters ...................................... 31
   1.18 Dates ..................................................... 33
   1.19 Accessing language info .................................. 33
   1.20 Hyphenation and line breaking ........................... 34
   1.21 Transforms ................................................ 36
   1.22 Selection based on BCP 47 tags ......................... 39
   1.23 Selecting scripts .......................................... 40
   1.24 Selecting directions ...................................... 40
   1.25 Language attributes ...................................... 44
   1.26 Hooks ..................................................... 44
   1.27 Languages supported by babel with ldf files ........ 46
   1.28 Unicode character properties in luatex ............... 47
   1.29 Tweaking some features .................................. 47
   1.30 Tips, workarounds, known issues and notes .......... 48
   1.31 Current and future work ................................ 49
   1.32 Tentative and experimental code ...................... 49

2. Loading languages with language.dat
   2.1 Format ..................................................... 50

3. The interface between the core of babel and the language definition files
   3.1 Guidelines for contributed languages .................... 52
   3.2 Basic macros ............................................... 52
   3.3 Skeleton .................................................. 53
   3.4 Support for active characters ........................... 54
   3.5 Support for saving macro definitions .................. 55
   3.6 Support for extending macros .......................... 55
   3.7 Macros common to a number of languages ............. 55
   3.8 Encoding-dependent strings ............................. 55
   3.9 Executing code based on the selector ................ 59

### II  Source code

4. Identification and loading of required files ............ 59

5. locale directory ............................................ 59
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 Tools</td>
<td>6.1 Multiple languages</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>6.2 The Package File (\texttt{fpX, babel.sty})</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>6.3 base</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>6.4 key=value options and other general options</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>6.5 Conditional loading of shorthands</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>6.6 Interlude for Plain</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>7 Multiple languages</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>7.1 Selecting the language</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>7.2 Errors</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>7.3 Hooks</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td>7.4 Setting up language files</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>7.5 Shorthands</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>7.6 Language attributes</td>
<td>95</td>
</tr>
<tr>
<td></td>
<td>7.7 Support for saving macro definitions</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>7.8 Short tags</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>7.9 Hyphens</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>7.10 Multicoding strings</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>7.11 Macros common to a number of languages</td>
<td>106</td>
</tr>
<tr>
<td></td>
<td>7.12 Making glyphs available</td>
<td>106</td>
</tr>
<tr>
<td></td>
<td>7.12.1 Quotation marks</td>
<td>106</td>
</tr>
<tr>
<td></td>
<td>7.12.2 Letters</td>
<td>107</td>
</tr>
<tr>
<td></td>
<td>7.12.3 Shorthands for quotation marks</td>
<td>108</td>
</tr>
<tr>
<td></td>
<td>7.12.4 Umlauts and tremas</td>
<td>109</td>
</tr>
<tr>
<td></td>
<td>7.13 Layout</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>7.14 Load engine specific macros</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td>7.15 Creating and modifying languages</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td>8 Adjusting the Babel behavior</td>
<td>131</td>
</tr>
<tr>
<td></td>
<td>8.1 Cross referencing macros</td>
<td>133</td>
</tr>
<tr>
<td></td>
<td>8.2 Marks</td>
<td>135</td>
</tr>
<tr>
<td></td>
<td>8.3 Preventing clashes with other packages</td>
<td>136</td>
</tr>
<tr>
<td></td>
<td>8.3.1 ifthen</td>
<td>136</td>
</tr>
<tr>
<td></td>
<td>8.3.2 varioref</td>
<td>137</td>
</tr>
<tr>
<td></td>
<td>8.3.3 hhline</td>
<td>137</td>
</tr>
<tr>
<td></td>
<td>8.4 Encoding and fonts</td>
<td>138</td>
</tr>
<tr>
<td></td>
<td>8.5 Basic bidi support</td>
<td>140</td>
</tr>
<tr>
<td></td>
<td>8.6 Local Language Configuration</td>
<td>143</td>
</tr>
<tr>
<td></td>
<td>8.7 Language options</td>
<td>143</td>
</tr>
<tr>
<td></td>
<td>9 The kernel of Babel (\texttt{babel.def, common})</td>
<td>146</td>
</tr>
<tr>
<td></td>
<td>10 Loading hyphenation patterns</td>
<td>147</td>
</tr>
<tr>
<td></td>
<td>11 Font handling with fontspec</td>
<td>151</td>
</tr>
<tr>
<td></td>
<td>12 Hooks for XeTeX and LuaTeX</td>
<td>155</td>
</tr>
<tr>
<td></td>
<td>12.1 XeTeX</td>
<td>155</td>
</tr>
<tr>
<td></td>
<td>12.2 Layout</td>
<td>157</td>
</tr>
<tr>
<td></td>
<td>12.3 LuaTeX</td>
<td>158</td>
</tr>
<tr>
<td></td>
<td>12.4 Southeast Asian scripts</td>
<td>164</td>
</tr>
<tr>
<td></td>
<td>12.5 CJK line breaking</td>
<td>165</td>
</tr>
<tr>
<td></td>
<td>12.6 Arabic justification</td>
<td>167</td>
</tr>
<tr>
<td></td>
<td>12.7 Common stuff</td>
<td>171</td>
</tr>
<tr>
<td></td>
<td>12.8 Automatic fonts and ids switching</td>
<td>171</td>
</tr>
<tr>
<td></td>
<td>12.9 Bidi</td>
<td>176</td>
</tr>
<tr>
<td></td>
<td>12.10 Layout</td>
<td>178</td>
</tr>
<tr>
<td></td>
<td>12.11 Lua: transforms</td>
<td>182</td>
</tr>
</tbody>
</table>
12.12 Lua: Auto bidi with basic and basic-r .............................. 190

13 Data for CJK 200

14 The 'nil' language 200

15 Support for Plain \TeX{} (plain.def) 201
   15.1 Not renaming hyphen.tex ......................................... 201
   15.2 Emulating some \TeX{} features .................................. 202
   15.3 General tools ..................................................... 202
   15.4 Encoding related macros ......................................... 206

16 Acknowledgements 209

Troubleshooting

Paragraph ended before \UTFviii@three@octets was complete ............... 5
No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the format ................................................................. 5
You are loading directly a language style .................................. 8
Unknown language ‘LANG’ ................................................... 8
Argument of \language@active@arg has an extra } ...................... 12
Package fontspec Warning: ‘Language ‘LANG’ not available for font ‘FONT’ with script ‘SCRIPT’ ‘Default’ language used instead’ ...................... 26
Package babel Info: The following fonts are not babel standard families ... 26
Part I
User guide

What is this document about? This user guide focuses on internationalization and localization with \LaTeX and pdflatex, xetex and luatex with the babel package. There are also some notes on its use with e-Plain and pdf-Plain \TeX. Part II describes the code, and usually it can be ignored.

What if I'm interested only in the latest changes? Changes and new features with relation to version 3.8 are highlighted with \texttt{New X.XX}, and there are some notes for the latest versions in the babel site. The most recent features can be still unstable.

Can I help? Sure! If you are interested in the \TeX multilingual support, please join the kadingira mail list. You can follow the development of babel in GitHub and make suggestions; feel free to fork it and make pull requests. If you are the author of a package, send to me a few test files which I'll add to mine, so that possible issues can be caught in the development phase.

It doesn't work for me! You can ask for help in some forums like tex.stackexchange, but if you have found a bug, I strongly beg you to report it in GitHub, which is much better than just complaining on an e-mail list or a web forum. Remember warnings are not errors by themselves, they just warn about possible problems or incompatibilities.

How can I contribute a new language? See section 3.1 for contributing a language.

I only need learn the most basic features. The first subsections (1.1-1.3) describe the traditional way of loading a language (with ldf files), which is usually all you need. The alternative way based on ini files, which complements the previous one (it does not replace it, although it is still necessary in some languages), is described below; go to 1.13.

I don't like manuals. I prefer sample files. This manual contains lots of examples and tips, but in GitHub there are many sample files.

1 The user interface

1.1 Monolingual documents

In most cases, a single language is required, and then all you need in \LaTeX is to load the package using its standard mechanism for this purpose, namely, passing that language as an optional argument. In addition, you may want to set the font and input encodings. Another approach is making the language a global option in order to let other packages detect and use it. This is the standard way in \LaTeX for an option – in this case a language – to be recognized by several packages. Many languages are compatible with xetex and luatex. With them you can use babel to localize the documents. When these engines are used, the Latin script is covered by default in current \LaTeX (provided the document encoding is UTF-8), because the font loader is preloaded and the font is switched to \texttt{lmroman}. Other scripts require loading fontspec. You may want to set the font attributes with fontspec, too.

EXAMPLE Here is a simple full example for “traditional” \TeX engines (see below for xetex and luatex). The packages fontenc and inputenc do not belong to babel, but they are included in the example because typically you will need them. It assumes UTF-8, the default encoding:

```latex
\documentclass{article}
\usepackage[T1]{fontenc}
```
\usepackage[french]{babel}
\begin{document}
Plus ça change, plus c'est la même chose!
\end{document}

Now consider something like:
\documentclass[french]{article}
\usepackage{babel}
\usepackage{varioref}

With this setting, the package varioref will also see the option french and will be able to use it.

EXAMPLE And now a simple monolingual document in Russian (text from the Wikipedia) with xetex or luatex. Note neither fontenc nor inputenc are necessary, but the document should be encoded in UTF-8 and a so-called Unicode font must be loaded (in this example \babelfont is used, described below).
\documentclass[russian]{article}
\usepackage{babel}
\babelfont{rm}{DejaVu Serif}
\begin{document}
Россия, находящаяся на пересечении множества культур, а также с учётом многонационального характера её населения, — отличается высокой степенью этнокультурного многообразия и способностью к межкультурному диалогу.
\end{document}

TROUBLESHOOTING A common source of trouble is a wrong setting of the input encoding. Depending on the \LaTeX version you can get the following somewhat cryptic error:

! Paragraph ended before \UTFviii@three@octets was complete.

Or the more explanatory:

! Package inputenc Error: Invalid UTF-8 byte ...

Make sure you set the encoding actually used by your editor.

NOTE Because of the way babel has evolved, “language” can refer to (1) a set of hyphenation patterns as preloaded into the format, (2) a package option, (3) an ldf file, and (4) a name used in the document to select a language or dialect. So, a package option refers to a language in a generic way – sometimes it is the actual language name used to select it, sometimes it is a file name loading a language with a different name, sometimes it is a file name loading several languages. Please, read the documentation for specific languages for further info.

TROUBLESHOOTING The following warning is about hyphenation patterns, which are not under the direct control of babel:
The document will be typeset, but very likely the text will not be correctly hyphenated. Some languages may be raising this warning wrongly (because they are not hyphenated); it is a bug to be fixed – just ignore it. See the manual of your distribution (Mac\TeX, Mi\TeX, Te\TeX Live, etc.) for further info about how to configure it.

**NOTE** With hyperref you may want to set the document language with something like:

```
\usepackage[pdflang=es-MX]{hyperref}
```

This is not currently done by babel and you must set it by hand.

**NOTE** Although it has been customary to recommend placing `\title`, `\author` and other elements printed by `\maketitle` after `\begin{document}`, mainly because of shorthands, it is advisable to keep them in the preamble. Currently there is no real need to use shorthands in those macros.

### 1.2 Multilingual documents

In multilingual documents, just use a list of the required languages as package or class options. The last language is considered the main one, activated by default. Sometimes, the main language changes the document layout (e.g., `spanish` and `french`).

**EXAMPLE** In \La\TeX, the preamble of the document:

```
\documentclass{article}
\usepackage[dutch,english]{babel}
```

would tell \La\TeX that the document would be written in two languages, Dutch and English, and that English would be the first language in use, and the main one.

You can also set the main language explicitly, but it is discouraged except if there is a real reason to do so:

```
\documentclass{article}
\usepackage[main=english,dutch]{babel}
```

Examples of cases where `main` is useful are the following.

**NOTE** Some classes load babel with a hardcoded language option. Sometimes, the main language can be overridden with something like that before `\documentclass`:

```
\PassOptionsToPackage{main=english}{babel}
```

**WARNING** Languages may be set as global and as package option at the same time, but in such a case you should set explicitly the main language with the package option `main`:

```
\documentclass[italian]{book}
\usepackage[ngerman,main=italian]{babel}
```

**WARNING** In the preamble the main language has not been selected, except hyphenation patterns and the name assigned to `\languagename` (in particular, shorthands, captions and date are not activated). If you need to define boxes and the like in the preamble, you might want to use some of the language selectors described below.
To switch the language there are two basic macros, described below in detail: \selectlanguage is used for blocks of text, while \foreignlanguage is for chunks of text inside paragraphs.

**EXAMPLE** A full bilingual document with pdftex follows. The main language is french, which is activated when the document begins. It assumes UTF-8:

```
\documentclass{article}
\usepackage[T1]{fontenc}
\usepackage[english,french]{babel}
\begin{document}
Plus ça change, plus c'est la même chose!
\selectlanguage{english}
And an English paragraph, with a short text in \foreignlanguage{french}{français}.
\end{document}
```

**EXAMPLE** With xetex and luatex, the following bilingual, single script document in UTF-8 encoding just prints a couple of ‘captions’ and \today in Danish and Vietnamese. No additional packages are required.

```
\documentclass{article}
\usepackage[vietnamese,danish]{babel}
\begin{document}
\prefacename{} -- \alsoname{} -- \today
\selectlanguage{vietnamese}
\prefacename{} -- \alsoname{} -- \today
\end{document}
```

**NOTE** Once loaded a language, you can select it with the corresponding BCP47 tag. See section 1.22 for further details.

### 1.3 Mostly monolingual documents

**New 3.39** Very often, multilingual documents consist of a main language with small pieces of text in another languages (words, idioms, short sentences). Typically, all you need is to set the line breaking rules and, perhaps, the font. In such a case, babel now does not require declaring these secondary languages explicitly, because the basic settings are loaded on the fly when the language is selected (and also when provided in the optional argument of \babelfont, if used.)

This is particularly useful, too, when there are short texts of this kind coming from an external source whose contents are not known on beforehand (for example, titles in a bibliography). At this regard, it is worth remembering that \babelfont does not load any font until required, so that it can be used just in case.

**EXAMPLE** A trivial document with the default font in English and Spanish, and FreeSerif in Russian is:
\documentclass[english]{article}
\usepackage{babel}
\babelfont[russian]{rm}{FreeSerif}
\begin{document}

English. \foreignlanguage{russian}{Русский}.
\foreignlanguage{spanish}{Español}.
\end{document}

NOTE Instead of its name, you may prefer to select the language with the corresponding BCP47 tag. This alternative, however, must be activated explicitly, because a two- or tree-letter word is a valid name for a language (eg, yi). See section 1.22 for further details.

1.4 Modifiers

New 3.9c The basic behavior of some languages can be modified when loading babel by means of modifiers. They are set after the language name, and are prefixed with a dot (only when the language is set as package option – neither global options nor the main key accepts them). An example is (spaces are not significant and they can be added or removed):\footnote{No predefined “axis” for modifiers are provided because languages and their scripts have quite different needs.}

\usepackage[latin.medieval, spanish.notilde.lcroman, danish]{babel}

Attributes (described below) are considered modifiers, ie, you can set an attribute by including it in the list of modifiers. However, modifiers are a more general mechanism.

1.5 Troubleshooting

• Loading directly sty files in \LaTeX (ie, \usepackage{⟨language⟩}) is deprecated and you will get the error:\footnote{In old versions the error read “You have used an old interface to call babel”, not very helpful.}

! Package babel Error: You are loading directly a language style.
(babel) This syntax is deprecated and you must use \usepackage[language]{babel}.

• Another typical error when using babel is the following:\footnote{In old versions the error read “You haven’t loaded the language LANG yet”.

! Package babel Error: Unknown language ‘#1’. Either you have
(babel) misspelled its name, it has not been installed,
(babel) or you requested it in a previous run. Fix its name,
(babel) install it or just rerun the file, respectively. In
(babel) some cases, you may need to remove the aux file

The most frequent reason is, by far, the latest (for example, you included spanish, but you realized this language is not used after all, and therefore you removed it from the option list). In most cases, the error vanishes when the document is typeset again, but in more severe ones you will need to remove the aux file.

1 No predefined “axis” for modifiers are provided because languages and their scripts have quite different needs.
2 In old versions the error read “You have used an old interface to call babel”, not very helpful.
3 In old versions the error read “You haven’t loaded the language LANG yet”. 
1.6 Plain
In e-Plain and pdf-Plain, load languages styles with `\input` and then use `\begindocument` (the latter is defined by babel):

```
\input estonian.sty
\begindocument
```

**WARNING** Not all languages provide a `sty` file and some of them are not compatible with those formats. Please, refer to Using babel with Plain for further details.

1.7 Basic language selectors
This section describes the commands to be used in the document to switch the language in multilingual documents. In most cases, only the two basic macros `\selectlanguage` and `\foreignlanguage` are necessary. The environments `otherlanguage`, `otherlanguage*`, and `hyphenrules` are auxiliary, and described in the next section.
The main language is selected automatically when the document environment begins.

`\selectlanguage{⟨language⟩}`
When a user wants to switch from one language to another he can do so using the macro `\selectlanguage`. This macro takes the language, defined previously by a language definition file, as its argument. It calls several macros that should be defined in the language definition files to activate the special definitions for the language chosen:

```
\selectlanguage{german}
```
This command can be used as environment, too.

**NOTE** For “historical reasons”, a macro name is converted to a language name without the leading \; in other words, `\selectlanguage{\german}` is equivalent to `\selectlanguage{german}`. Using a macro instead of a “real” name is deprecated. [New 3.43] However, if the macro name does not match any language, it will get expanded as expected.

**NOTE** Bear in mind `\selectlanguage` can be automatically executed, in some cases, in the auxiliary files, at heads and foots, and after the environment `otherlanguage*`.

**WARNING** If used inside braces there might be some non-local changes, as this would be roughly equivalent to:

```
{\selectlanguage{⟨inner-language⟩} ...}\selectlanguage{⟨outer-language⟩}
```
If you want a change which is really local, you must enclose this code with an additional grouping level.

**WARNING** There are a couple of issues related to the way the language information is written to the auxiliary files:

- `\selectlanguage` should not be used inside some boxed environments (like floats or `minipage`) to switch the language if you need the information written to the aux be correctly synchronized. This rarely happens, but if it were the case, you must use `otherlanguage` instead.
- In addition, this macro inserts a `\write` in vertical mode, which may break the vertical spacing in some cases (for example, between lists). [New 3.64] The behavior can be adjusted with `\b babeladjust{select.write=(mode)}`, where `(mode)` is shift (which shifts the skips down and adds a `\penalty`); keep (the default – with it the `\write` and the skips are kept in the order they are written), and omit (which may seem a too drastic solution, because nothing is written, but more often than not this command is applied to more or less short texts with no sectioning or similar commands and therefore no language synchronization is necessary).
\texttt{\textbackslash foreignlanguage} \{\texttt{(option-list)}\}\{\texttt{language}\}\{\texttt{text}\}

The command \texttt{foreignlanguage} takes two arguments; the second argument is a phrase to be typeset according to the rules of the language named in its first one. This command (1) only switches the extra definitions and the hyphenation rules for the language, not the names and dates, (2) does not send information about the language to auxiliary files (i.e., the surrounding language is still in force), and (3) it works even if the language has not been set as package option (but in such a case it only sets the hyphenation patterns and a warning is shown). With the bidi option, it also enters in horizontal mode (this is not done always for backwards compatibility), and since it is meant for phrases only the text direction (and not the paragraph one) is set.

\textbf{New 3.44} As already said, captions and dates are not switched. However, with the optional argument you can switch them, too. So, you can write:

\texttt{\textbackslash foreignlanguage[\texttt{date}}\{\texttt{polish}\}\{\texttt{\today}\}

In addition, captions can be switched with captions (or both, of course, with date, captions). Until 3.43 you had to write something like \texttt{\selectlanguage{..} ..}, which was not always the most convenient way.

\subsection*{1.8 Auxiliary language selectors}

\texttt{\textbackslash begin\{otherlanguage\} \{\texttt{language}\} \ldots \textbackslash end\{otherlanguage\}}

The environment \texttt{otherlanguage} does basically the same as \texttt{selectlanguage}, except that language change is (mostly) local to the environment. Actually, there might be some non-local changes, as this environment is roughly equivalent to:

\begin{verbatim}
\begingroup
\selectlanguage{<inner-language>}
\selectlanguage{<outer-language>}
\end{verbatim}

If you want a change which is really local, you must enclose this environment with an additional grouping, like braces \{\}. Spaces after the environment are ignored.

\texttt{\textbackslash begin\{otherlanguage\}*}\{\texttt{(option-list)}\}\{\texttt{language}\}\ldots\texttt{\textbackslash end\{otherlanguage\}*}

Same as \texttt{foreignlanguage} but as environment. Spaces after the environment are not ignored. This environment was originally intended for intermixing left-to-right typesetting with right-to-left typesetting in engines not supporting a change in the writing direction inside a line. However, by default it never complied with the documented behavior and it is just a version as environment of \texttt{foreignlanguage}, except when the option bidi is set – in this case, \texttt{foreignlanguage} emits a \texttt{\leavevmode}, while \texttt{otherlanguage\*} does not.

\subsection*{1.9 More on selection}

\texttt{\textbackslash babeltags} \{\texttt{\textbackslash babeltags}\} \{\texttt{\{tag1\} = \{language1\}, \{tag2\} = \{language2\}, ...}\}

\textbf{New 3.9i} In multilingual documents with many language-switches the commands above can be cumbersome. With this tool shorter names can be defined. It adds nothing really new – it is just syntactical sugar.
It defines \text{⟨tag1⟩}{⟨text⟩} to be \foreignlanguage{⟨language1⟩}{⟨text⟩}, and \begin{⟨tag1⟩} to be \begin{otherlanguage*}{⟨language1⟩}, and so on. Note \⟨tag1⟩ is also allowed, but remember to set it locally inside a group.

**WARNING** There is a clear drawback to this feature, namely, the ‘prefix’ \text... is heavily overloaded in \LaTeX{} and conflicts with existing macros may arise (\textlatin, \textbar{}, \textit, \textcolor and many others). The same applies to environments, because arabic conflicts with \arabic. Furthermore, and because of this overloading, detecting the language of a chunk of text by external tools can become unfeasible. Except if there is a reason for this ‘syntactical sugar’, the best option is to stick to the default selectors or to define your own alternatives.

**EXAMPLE** With

\begin{verbatim}
\babeltags{de = german}
\end{verbatim}

you can write

\begin{verbatim}
text \textde{German text} text \end{verbatim}

and

\begin{verbatim}
text \begin{de} German text \end{de}
\end{verbatim}

**NOTE** Something like \babeltags{finnish = finnish} is legitimate – it defines \textfinnish and \finnish (and, of course, \begin{finnish}).

**NOTE** Actually, there may be another advantage in the ‘short’ syntax \text{⟨tag⟩}, namely, it is not affected by \MakeUppercase (while \foreignlanguage is).

**\babelensure** [include=⟨commands⟩, exclude=⟨commands⟩, fontenc=⟨encoding⟩] {⟨language⟩}

New 3.9i: Except in a few languages, like russian, captions and dates are just strings, and do not switch the language. That means you should set it explicitly if you want to use them, or hyphenation (and in some cases the text itself) will be wrong. For example:

\begin{verbatim}
\foreignlanguage{russian}{text \foreignlanguage{polish}{\seename} text}
\end{verbatim}

Of course, \TeX{} can do it for you. To avoid switching the language all the while, \babelensure redefines the captions for a given language to wrap them with a selector:

\begin{verbatim}
\babelensure{polish}
\end{verbatim}

By default only the basic captions and \today are redefined, but you can add further macros with the key include in the optional argument (without commas). Macros not to be modified are listed in exclude. You can also enforce a font encoding with the option fontenc.\footnote{With it, encoded strings may not work as expected.} A couple of examples:

\begin{verbatim}
\babelensure[include=\Today]{spanish}
\babelensure[fontenc=T5]{vietnamese}
\end{verbatim}

They are activated when the language is selected (at the afterextras event), and it makes some assumptions which could not be fulfilled in some languages. Note also you should include only macros defined by the language, not global macros (eg, \TeX{} of \dag). With ini files (see below), captions are ensured by default.
1.10 Shorthands

A shorthand is a sequence of one or two characters that expands to arbitrary \TeX code. Shorthands can be used for different kinds of things; for example: (1) in some languages shorthands such as "a are defined to be able to hyphenate the word if the encoding is OT1; (2) in some languages shorthands such as \ are used to insert the right amount of white space; (3) several kinds of discretionary and breaks can be inserted easily with "\-, ", etc.

The package inputenc as well as xetex and luatex have alleviated entering non-ASCII characters, but minority languages and some kinds of text can still require characters not directly available on the keyboards (and sometimes not even as separated or precomposed Unicode characters). As to the point 2, now pdfTeX provides \kncode, and luatex can manipulate the glyph list. Tools for point 3 can be still very useful in general.

There are four levels of shorthands: user, language, system, and language user (by order of precedence). In most cases, you will use only shorthands provided by languages.

NOTE Keep in mind the following:

1. Activated chars used for two-char shorthands cannot be followed by a closing brace } and the spaces following are gobbled. With one-char shorthands (eg, :) they are preserved.
2. If on a certain level (system, language, user, language user) there is a one-char shorthand, two-char ones starting with that char and on the same level are ignored.
3. Since they are active, a shorthand cannot contain the same character in its definition (except if deactivated with, eg, \string).

TROUBLESHOOTING A typical error when using shorthands is the following:

! Argument of \language@active@arg has an extra }.

It means there is a closing brace just after a shorthand, which is not allowed (eg, "). Just add } after (eg, ")}).

\shorthandon \shorthandoff

\{⟨shorthands-list⟩\}
\*{⟨shorthands-list⟩}

It is sometimes necessary to switch a shorthand character off temporarily, because it must be used in an entirely different way. For this purpose, the user commands \shorthandoff and \shorthandon are provided. They each take a list of characters as their arguments. The command \shorthandoff sets the \catcode for each of the characters in its argument to other (12); the command \shorthandon sets the \catcode to active (13). Both commands only work on ‘known’ shorthand characters.

New 3.9a However, \shorthandoff does not behave as you would expect with characters like ~ or ^, because they usually are not “other”. For them \shorthandoff* is provided, so that with

\shorthandoff*{\-^}

~ is still active, very likely with the meaning of a non-breaking space, and ^ is the superscript character. The catcodes used are those when the shorthands are defined, usually when language files are loaded.

If you do not need shorthands, or prefer an alternative approach of your own, you may want to switch them off with the package option shorthands=off, as described below.

WARNING It is worth emphasizing these macros are meant for temporary changes. Whenever possible and if there are not conflicts with other packages, shorthands must be always enabled (or disabled).
The command \useshorthands initiates the definition of user-defined shorthand sequences. It has one argument, the character that starts these personal shorthands.

New 3.9a User shorthands are not always alive, as they may be deactivated by languages (for example, if you use " for your user shorthands and switch from german to french, they stop working). Therefore, a starred version \useshorthands*{⟨char⟩} is provided, which makes sure shorthands are always activated.

Currently, if the package option shorthands is used, you must include any character to be activated with \useshorthands. This restriction will be lifted in a future release.

The command \defineshorthand takes two arguments: the first is a one- or two-character shorthand sequence, and the second is the code the shorthand should expand to.

New 3.9a An optional argument allows to (re)define language and system shorthands (some languages do not activate shorthands, so you may want to add \languageshorthands{⟨lang⟩} to the corresponding \extras{lang}, as explained below). By default, user shorthands are (re)defined.

User shorthands override language ones, which in turn override system shorthands. Language-dependent user shorthands (new in 3.9) take precedence over "normal" user shorthands.

**EXAMPLE** Let’s assume you want a unified set of shorthand for discretionary languages do not define shorthands consistently, and "-, \-, -" have different meanings). You can start with, say:

\useshorthands*{"}
\defineshorthand{"\{\babelhyphen{soft}\}
\defineshorthand{"\{\babelhyphen{hard}\}

However, the behavior of hyphens is language-dependent. For example, in languages like Polish and Portuguese, a hard hyphen inside compound words are repeated at the beginning of the next line. You can then set:

\defineshorthand{\{\lang\}="\{\babelhyphen{repeat}\}

Here, options with * set a language-dependent user shorthand, which means the generic one above only applies for the rest of languages; without * they would (re)define the language shorthands instead, which are overridden by user ones.

Now, you have a single unified shorthand ("-), with a content-based meaning (`compound word hyphen`) whose visual behavior is that expected in each context.

The command \languageshorthands can be used to switch the shorthands on the language level. It takes one argument, the name of a language or none (the latter does what its name suggests). Note that for this to work the language should have been specified as an option when loading the babel package. For example, you can use in english the shorthands defined by ngerman with

\addto\extrasenglish{\languageshorthands{ngerman}}

(You may also need to activate them as user shorthands in the preamble with, for example, \useshorthands or \useshorthands*)

---

5 Actually, any name not corresponding to a language group does the same as none. However, follow this convention because it might be enforced in future releases of babel to catch possible errors.
Very often, this is a more convenient way to deactivate shorthands than \shorthandoff, for example if you want to define a macro to easy typing phonetic characters with tipa:

\newcommand{\myipa}[1]{\languageshorthands{none}\tipaencoding#1}

With this command you can use a shorthand even if (1) not activated in\shorthand{}s (in this case only shorthands for the current language are taken into account, i.e., not user shorthands), (2) turned off with \shorthandoff or (3) deactivated with the internal \bbl@deactivate; for example, \babelshorthand{"u} or \babelshorthand{:}. (You can conveniently define your own macros, or even your own user shorthands provided they do not overlap.)

Since by default shorthands are not activated until \begin{document}, you may use this macro when defining the \title{} in the preamble:

\title{Documento científico\babelshorthand{"--} técnico}

For your records, here is a list of shorthands, but you must double check them, as they may change:

Languages with no shorthands  Croatian, English (any variety), Indonesian, Hebrew, Interlingua, Irish, Lower Sorbian, Malaysian, North Sami, Romanian, Scottish, Welsh

Languages with only " as defined shorthand character  Albanian, Bulgarian, Danish, Dutch, Finnish, German (old and new orthography, also Austrian), Icelandic, Italian, Norwegian, Polish, Portuguese (also Brazilian), Russian, Serbian (with Latin script), Slovene, Swedish, Ukrainian, Upper Sorbian

Basque " ' ~
Breton : ; ? !
Catalan " ' 
Czech " -
Esperanto ^
Estonian " ~
French (all varieties): ; ? !
Galician " . ' ~ < >
Greek ~
Hungarian \\nKurmanji ^
Latin " ^ =
Slovak " ^ ' -
Spanish " . < > ' ~
Turkish : ! =

In addition, the babel core declares ~ as a one-char shorthand which is let, like the standard ~, to a non-breaking space.

\ifbabelshorthand {\character}{\true}{\false}

New 3.23 Tests if a character has been made a shorthand.

\aliasshorthand {\original}{\alias}

The command \aliasshorthand can be used to let another character perform the same functions as the default shorthand character. If one prefers for example to use the

\footnote{Thanks to Enrico Gregorio}

\footnote{This declaration serves to nothing, but it is preserved for backward compatibility.}
character / over " in typing Polish texts, this can be achieved by entering \aliashorthand{"}{/}. For the reasons in the warning below, usage of this macro is not recommended.

NOTE The substitute character must not have been declared before as shorthand (in such a case, \aliashortands is ignored).

EXAMPLE The following example shows how to replace a shorthand by another

\aliashorthand{-}{^}
\AtBeginDocument{\shorthandoff{-}}

WARNING Shorthands remember somehow the original character, and the fallback value is that of the latter. So, in this example, if no shorthand if found, ^ expands to a non-breaking space, because this is the value of ~ (internally, ^ still calls \active@char~ or \normal@char~). Furthermore, if you change the system value of ^ with \defineshorthand nothing happens.

1.11 Package options

New 3.9a These package options are processed before language options, so that they are taken into account irrespective of its order. The first three options have been available in previous versions.

KeepShorthandsActive Tells babel not to deactivate shorthands after loading a language file, so that they are also available in the preamble.

activeacute For some languages babel supports this options to set ' as a shorthand in case it is not done by default.

activegrave Same for `.

shorthands= (char\char)... | off
The only language shorthands activated are those given, like, eg:

\usepackage[esperanto,french,shorthands=;!?]{babel}

If ' is included, activeacute is set; if ` is included, activegrave is set. Active characters (like ~) should be preceded by \string (otherwise they will be expanded by \IfX before they are passed to the package and therefore they will not be recognized); however, t is provided for the common case of ~ (as well as c for not so common case of the comma). With shorthands=off no language shorthands are defined. As some languages use this mechanism for tools not available otherwise, a macro \babelshorthand is defined, which allows using them; see above.

safe= none | ref | bib
Some \IfX macros are redefined so that using shorthands is safe. With safe=bib only \nocite, \bibcite and \bibitem are redefined. With safe=ref only \newlabel, \ref and \pageref are redefined (as well as a few macros from varioref and ifthen). With safe=none no macro is redefined. This option is strongly recommended, because a good deal of incompatibilities and errors are related to these redefinitions. As of New 3.34, in \IfX based engines (ie, almost every engine except the oldest ones) shorthands can be used in these macros (formerly you could not).

math= active | normal
Shorthands are mainly intended for text, not for math. By setting this option with the value normal they are deactivated in math mode (default is active) and things like ${a'}$ (a closing brace after a shorthand) are not a source of trouble anymore.
config=⟨file⟩
Load ⟨file⟩.cfg instead of the default config file bblopts.cfg (the file is loaded even with noconfigs).

main=⟨language⟩
Sets the main language, as explained above, ie, this language is always loaded last. If it is not given as package or global option, it is added to the list of requested languages.

headfoot=⟨language⟩
By default, headlines and footlines are not touched (only marks), and if they contain language-dependent macros (which is not usual) there may be unexpected results. With this option you may set the language in heads and foots.

noconfigs
Global and language default config files are not loaded, so you can make sure your document is not spoilt by an unexpected .cfg file. However, if the key config is set, this file is loaded.

showlanguages
Prints to the log the list of languages loaded when the format was created: number (remember dialects can share it), name, hyphenation file and exceptions file.

noselect
New 3.9l Language settings for uppercase and lowercase mapping (as set by \SetCase) are ignored. Use only if there are incompatibilities with other packages.

silent
New 3.9l No warnings and no infos are written to the log file.\footnote{You can use alternatively the package silence.}

strings=generic | unicode | encoded | ⟨label⟩ | ⟨font encoding⟩
Selects the encoding of strings in languages supporting this feature. Predefined labels are generic (for traditional \TeX, \LICR and ASCII strings), unicode (for engines like xetex and \luatex) and encoded (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in \MakeUppercase and the like (this feature misuses some internal \LaTeX tools, so use it only as a last resort).

hyphenmap=off | first | select | other | other*
New 3.9g Sets the behavior of case mapping for hyphenation, provided the language defines it.\footnote{Turned off in plain.} It can take the following values:

off deactivate this feature and no case mapping is applied;
first sets it at the first switching commands in the current or parent scope (typically, when the aux file is first read and at \begin{document}, but also the first \selectlanguage in the preamble), and it’s the default if a single language option has been stated;\footnote{Duplicated options count as several ones.}
select sets it only at \selectlanguage;
other also sets it at otherlanguage;
other* also sets it at otherlanguage* as well as in heads and foots (if the option headfoot is used) and in auxiliary files (ie, at \select@language), and it’s the default if several language options have been stated. The option first can be regarded as an optimized version of other* for monolingual documents.\footnote{Providing foreign is pointless, because the case mapping applied is that at the end of the paragraph, but if either xetex or \luatex change this behavior it might be added. On the other hand, other is provided even if I [JBL] think it isn’t really useful, but who knows.}
1.12 The base option

With this package option babel just loads some basic macros (those in switch.def), defines \AfterBabelLanguage and exits. It also selects the hyphenation patterns for the last language passed as option (by its name in language.dat). There are two main uses: classes and packages, and as a last resort in case there are, for some reason, incompatible languages. It can be used if you just want to select the hyphenation patterns of a single language, too.

\AfterBabelLanguage \{\langle option-name\}\}\{\langle code\}\}

This command is currently the only provided by base. Executes \langle code\rangle when the file loaded by the corresponding package option is finished (at \ldf@finish). The setting is global. So

\AfterBabelLanguage\{french\}\{...\}

does ... at the end of french.ldf. It can be used in ldf files, too, but in such a case the code is executed only if \langle option-name\rangle is the same as \CurrentOption (which could not be the same as the option name as set in \usepackage!).

EXAMPLE Consider two languages foo and bar defining the same \macro with \newcommand. An error is raised if you attempt to load both. Here is a way to overcome this problem:

\usepackage[base]{babel}
\AfterBabelLanguage\{foo\}\{% \let\macroFoo\macro \let\macro\relax \usepackage[foo,bar]{babel}

NOTE With a recent version of \LaTeX, an alternative method to execute some code just after an ldf file is loaded is with \AddToHook and the hook file/<language>.ldf/after. Babel does not predeclare it, and you have to do it yourself with \ActivateGenericHook.

WARNING Currently this option is not compatible with languages loaded on the fly.

1.13 ini files

An alternative approach to define a language (or, more precisely, a locale) is by means of an ini file. Currently babel provides about 250 of these files containing the basic data required for a locale, plus basic templates for 500 about locales. ini files are not meant only for babel, and they has been devised as a resource for other packages. To easy interoperability between \LaTeX and other systems, they are identified with the BCP 47 codes as preferred by the Unicode Common Locale Data Repository, which was used as source for most of the data provided by these files, too (the main exception being the \...name strings).

Most of them set the date, and many also the captions (Unicode and LICR). They will be evolving with the time to add more features (something to keep in mind if backward
compatibility is important). The following section shows how to make use of them by means of `\babelprovide`. In other words, `\babelprovide` is mainly meant for auxiliary tasks, and as alternative when the `ldf`, for some reason, does work as expected.

**EXAMPLE** Although Georgian has its own `ldf` file, here is how to declare this language with an ini file in Unicode engines.

```
\documentclass{book}

\usepackage{babel}
\babelprovide[import, main]{georgian}
\babelfont{rm}{Renderer=Harfbuzz}{DejaVu Sans}

\begin{document}
\tableofcontents
\chapter{სამზარეულო და სუფრის ტრადიციები}
ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.
\end{document}
```

**New 3.49** Alternatively, you can tell babel to load all or some languages passed as options with `\babelprovide` and not from the `ldf` file in a few few typical cases. Thus, `provide=*` means 'load the main language with the `\babelprovide` mechanism instead of the `ldf` file' applying the basic features, which in this case means `import, main`. There are (currently) three options:

- `provide=*` is the option just explained, for the main language;
- `provide+[*` is the same for additional languages (the main language is still the `ldf` file);
- `provide+[*` is the same for all languages, ie, main and additional.

**EXAMPLE** The preamble in the previous example can be more compactly written as:

```
\documentclass{book}
\usepackage[georgian, provide=*]{babel}
\babelfont{rm}{Renderer=Harfbuzz}{DejaVu Sans}
```

Or also:

```
\documentclass[georgian]{book}
\usepackage[provide=*]{babel}
\babelfont{rm}{Renderer=Harfbuzz}{DejaVu Sans}
```

**NOTE** The ini files just define and set some parameters, but the corresponding behavior is not always implemented. Also, there are some limitations in the engines. A few remarks follow (which could no longer be valid when you read this manual, if the packages involved han been updated). The Harfbuzz renderer has still some issues, so as a rule of thumb prefer the default renderer, and resort to Harfbuzz only if the former does not work for you. Fortunately, fonts can be loaded twice with different renderers; for example:

```
\babelfont[spanish]{rm}{FreeSerif}
\babelfont[hindi]{rm}{Renderer=Harfbuzz}{FreeSerif}
```
Arabic  Monolingual documents mostly work in luatex, but it must be fine tuned, particularly math and graphical elements like picture. In xetex babel resorts to the bidi package, which seems to work.

Hebrew  Niqqud marks seem to work in both engines, but depending on the font cantillation marks might be misplaced (xetex or luatex with Harfbuzz seems better).

Devanagari  In luatex and the the default renderer many fonts work, but some others do not, the main issue being the ‘ra’. You may need to set explicitly the script to either deva or dev2, eg:

\newfontscript{Devanagari}{deva}

Other Indic scripts are still under development in the default luatex renderer, but should work with Renderer=Harfbuzz. They also work with xetex, although unlike with luatex fine tuning the font behavior is not always possible.

Southeast scripts  Thai works in both luatex and xetex, but line breaking differs (rules are hard-coded in xetex, but they can be modified in luatex). Lao seems to work, too, but there are no patterns for the latter in luatex. Khamer clusters are rendered wrongly with the default renderer. The comment about Indic scripts and luatex also applies here. Some quick patterns can help, with something similar to:

\babelprovide[import, hyphenrules=+]{lao}
\babelpatterns[lao]{1ດ 1ມ 1ອ 1ງ 1ກ 1າ} % Random

East Asia scripts  Settings for either Simplified of Traditional should work out of the box, with basic line breaking with any renderer. Although for a few words and short texts the ini files should be fine, CJK texts are best set with a dedicated framework (CJK, luatexja, kotex, CTeX, etc.). This is what the class ltjbook does with luatex, which can be used in conjunction with the 1df for japanese, because the following piece of code loads luatexja:

\documentclass[japanese]{ltjbook}
\usepackage{babel}

Latin, Greek, Cyrillic  Combining chars with the default luatex font renderer might be wrong; on the other hand, with the Harfbuzz renderer diacritics are stacked correctly, but many hyphenations points are discarded (this bug is related to kerning, so it depends on the font). With xetex both combining characters and hyphenation work as expected (not quite, but in most cases it works; the problem here are font clusters).

NOTE  Wikipedia defines a locale as follows: “In computing, a locale is a set of parameters that defines the user’s language, region and any special variant preferences that the user wants to see in their user interface. Usually a locale identifier consists of at least a language code and a country/region code.” Babel is moving gradually from the old and fuzzy concept of language to the more modern of locale. Note each locale is by itself a separate “language”, which explains why there are so many files. This is on purpose, so that possible variants can be created and/or redefined easily.

Here is the list (u means Unicode captions, and l means LICR captions):

<table>
<thead>
<tr>
<th>Code</th>
<th>Language</th>
<th>Code</th>
<th>Language</th>
<th>Code</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>af</td>
<td>Afrikaans</td>
<td>bem</td>
<td>Bemba</td>
<td></td>
<td></td>
</tr>
<tr>
<td>agq</td>
<td>Aghem</td>
<td>bez</td>
<td>Bena</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ak</td>
<td>Akan</td>
<td>bg</td>
<td>Bulgarian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>am</td>
<td>Amharic</td>
<td>bm</td>
<td>Bambara</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ar</td>
<td>Arabic</td>
<td>bn</td>
<td>Bangla</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ar-DZ</td>
<td>Arabic</td>
<td>bo</td>
<td>Tibetan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ar-MA</td>
<td>Arabic</td>
<td>brx</td>
<td>Bodo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ar-SY</td>
<td>Arabic</td>
<td>bs-Cyr</td>
<td>Bosnian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>as</td>
<td>Assamese</td>
<td>bs-Lat</td>
<td>Bosnian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>asa</td>
<td>Asu</td>
<td>bs</td>
<td>Bosnian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ast</td>
<td>Asturian</td>
<td>ca</td>
<td>Catalan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>az-Cyr</td>
<td>Azerbaijani</td>
<td>ce</td>
<td>Chechen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>az-Lat</td>
<td>Azerbaijani</td>
<td>cgg</td>
<td>Chiga</td>
<td></td>
<td></td>
</tr>
<tr>
<td>az</td>
<td>Azerbaijani</td>
<td>chr</td>
<td>Cherokee</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bas</td>
<td>Basaa</td>
<td>ckb</td>
<td>Central Kurdish</td>
<td></td>
<td></td>
</tr>
<tr>
<td>be</td>
<td>Belarusian</td>
<td>cop</td>
<td>Coptic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Language</td>
<td>Code</td>
<td>Language</td>
<td>Code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
<td>----------</td>
<td>------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cs</td>
<td>cs</td>
<td>cu</td>
<td>cu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cu-Cyrs</td>
<td>cu-Cyrs</td>
<td>cu-Glag</td>
<td>cu-Glag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cu-Glag</td>
<td>cu-Glag</td>
<td>cy</td>
<td>cy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>da</td>
<td>da</td>
<td>de</td>
<td>de</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dav</td>
<td>dav</td>
<td>de-AT</td>
<td>de-AT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>de-CH</td>
<td>de-CH</td>
<td>de</td>
<td>de</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dje</td>
<td>dje</td>
<td>dz</td>
<td>dz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dua</td>
<td>dua</td>
<td>ebu</td>
<td>ebu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ee</td>
<td>ee</td>
<td>el</td>
<td>el</td>
<td></td>
<td></td>
</tr>
<tr>
<td>el-polyton</td>
<td>el-polyton</td>
<td>en-AU</td>
<td>en-AU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>en-CA</td>
<td>en-CA</td>
<td>en-GB</td>
<td>en-GB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>en-NZ</td>
<td>en-NZ</td>
<td>en-US</td>
<td>en-US</td>
<td></td>
<td></td>
</tr>
<tr>
<td>en</td>
<td>en</td>
<td>eo</td>
<td>eo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>es-MX</td>
<td>es-MX</td>
<td>es</td>
<td>es</td>
<td></td>
<td></td>
</tr>
<tr>
<td>et</td>
<td>et</td>
<td>eu</td>
<td>eu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ewo</td>
<td>ewo</td>
<td>fa</td>
<td>fa</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ff</td>
<td>ff</td>
<td>fi</td>
<td>fi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fil</td>
<td>fil</td>
<td>fo</td>
<td>fo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr</td>
<td>fr</td>
<td>fr-BE</td>
<td>fr-BE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr-CA</td>
<td>fr-CA</td>
<td>fr-CH</td>
<td>fr-CH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr-LU</td>
<td>fr-LU</td>
<td>fur</td>
<td>fur</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fy</td>
<td>fy</td>
<td>ga</td>
<td>ga</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gd</td>
<td>gd</td>
<td>gl</td>
<td>gl</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grc</td>
<td>grc</td>
<td>gsw</td>
<td>gsw</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gu</td>
<td>gu</td>
<td>guz</td>
<td>guz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gv</td>
<td>gv</td>
<td>ha-GH</td>
<td>ha-GH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ha-NE</td>
<td>ha-NE</td>
<td>ha</td>
<td>ha</td>
<td></td>
<td></td>
</tr>
<tr>
<td>haw</td>
<td>haw</td>
<td>he</td>
<td>he</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hi</td>
<td>hi</td>
<td>hr</td>
<td>hr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hu</td>
<td>hu</td>
<td>hy</td>
<td>hy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>id</td>
<td>ia</td>
<td>ia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>id-polyton</td>
<td>id-polyton</td>
<td>ig</td>
<td>ig</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ig</td>
<td>ig</td>
<td>is</td>
<td>is</td>
<td></td>
<td></td>
</tr>
<tr>
<td>it</td>
<td>it</td>
<td>ja</td>
<td>ja</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jgo</td>
<td>jgo</td>
<td>jmc</td>
<td>jmc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ka</td>
<td>ka</td>
<td>kab</td>
<td>kab</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kde</td>
<td>kde</td>
<td>kea</td>
<td>kea</td>
<td></td>
<td></td>
</tr>
<tr>
<td>khq</td>
<td>khq</td>
<td>kikyuyu</td>
<td>kikyuyu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ko</td>
<td>ko</td>
<td>konkani</td>
<td>konkani</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ks</td>
<td>ks</td>
<td>Kashmiri</td>
<td>Kashmiri</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ksb</td>
<td>ksb</td>
<td>Shambala</td>
<td>Shambala</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ksf</td>
<td>ksf</td>
<td>Colognian</td>
<td>Colognian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ksh</td>
<td>ksh</td>
<td>Cornish</td>
<td>Cornish</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ky</td>
<td>ky</td>
<td>Kyrgyz</td>
<td>Kyrgyz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lag</td>
<td>lag</td>
<td>Langi</td>
<td>Langi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lb</td>
<td>lb</td>
<td>Luxembourgish</td>
<td>Luxembourgish</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lg</td>
<td>lg</td>
<td>Ganda</td>
<td>Ganda</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lkt</td>
<td>lkt</td>
<td>Lakota</td>
<td>Lakota</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ln</td>
<td>ln</td>
<td>Lingala</td>
<td>Lingala</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lo</td>
<td>lo</td>
<td>Lao</td>
<td>Lao</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lrc</td>
<td>lrc</td>
<td>Northern Luri</td>
<td>Northern Luri</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lt</td>
<td>lt</td>
<td>Lithuanian</td>
<td>Lithuanian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lu</td>
<td>lu</td>
<td>Luo</td>
<td>Luo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>luy</td>
<td>luy</td>
<td>Luyia</td>
<td>Luyia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lv</td>
<td>lv</td>
<td>Latvian</td>
<td>Latvian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mas</td>
<td>mas</td>
<td>Masai</td>
<td>Masai</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mer</td>
<td>mer</td>
<td>Meru</td>
<td>Meru</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mfe</td>
<td>mfe</td>
<td>Morisyen</td>
<td>Morisyen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mg</td>
<td>mg</td>
<td>Malagasy</td>
<td>Malagasy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mgh</td>
<td>mgh</td>
<td>Makhulu-Meeto</td>
<td>Makhulu-Meeto</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mgo</td>
<td>mgo</td>
<td>Meta'</td>
<td>Meta'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mk</td>
<td>mk</td>
<td>Macedonian</td>
<td>Macedonian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ml</td>
<td>ml</td>
<td>Malayalam</td>
<td>Malayalam</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mn</td>
<td>mn</td>
<td>Mongolian</td>
<td>Mongolian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mr</td>
<td>mr</td>
<td>Marathi</td>
<td>Marathi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ms-BN</td>
<td>ms-BN</td>
<td>ms-SG</td>
<td>ms-SG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ms</td>
<td>ms</td>
<td>Malay</td>
<td>Malay</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ms</td>
<td>ms</td>
<td>Malay'</td>
<td>Malay'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

20
In some contexts (currently `\babelfont`) an ini file may be loaded by its name. Here is the list of the names currently supported. With these languages, `\babelfont` loads (if not done before) the language and script names (even if the language is defined as a package option)
with an ldf file). These are also the names recognized by \babel\provide with a valueless import.

<table>
<thead>
<tr>
<th>Language</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>aghem</td>
<td>chinese-hans-mo</td>
</tr>
<tr>
<td>akan</td>
<td>chinese-hans-sg</td>
</tr>
<tr>
<td>albanian</td>
<td>chinese-hans</td>
</tr>
<tr>
<td>american</td>
<td>chinese-hant-hk</td>
</tr>
<tr>
<td>amharic</td>
<td>chinese-hant-mo</td>
</tr>
<tr>
<td>ancientgreek</td>
<td>chinese-hant</td>
</tr>
<tr>
<td>arabic</td>
<td>chinese-simplified-hongkongsarchina</td>
</tr>
<tr>
<td>arabic-algeria</td>
<td>chinese-simplified-macausarchina</td>
</tr>
<tr>
<td>arabic-DZ</td>
<td>chinese-simplified-singapore</td>
</tr>
<tr>
<td>arabic-morocco</td>
<td>chinese-simplified</td>
</tr>
<tr>
<td>arabic-MA</td>
<td>chinese-traditional-hongkongsarchina</td>
</tr>
<tr>
<td>arabic-syria</td>
<td>chinese-traditional-macausarchina</td>
</tr>
<tr>
<td>arabic-SY</td>
<td>chinese-traditional</td>
</tr>
<tr>
<td>armenian</td>
<td>chinese</td>
</tr>
<tr>
<td>assamese</td>
<td>churchslavic</td>
</tr>
<tr>
<td>asturian</td>
<td>churchslavic-cyrs</td>
</tr>
<tr>
<td>asu</td>
<td>churchslavic-oldcyrillic12</td>
</tr>
<tr>
<td>austrian</td>
<td>churchslavic-glag</td>
</tr>
<tr>
<td>azerbaijani-cyrillic</td>
<td>colognian</td>
</tr>
<tr>
<td>azerbaijani-cyril</td>
<td>cornish</td>
</tr>
<tr>
<td>azerbaijani-latin</td>
<td>croatian</td>
</tr>
<tr>
<td>azerbaijani-latn</td>
<td>czech</td>
</tr>
<tr>
<td>azerbaijani</td>
<td>danish</td>
</tr>
<tr>
<td>baafia</td>
<td>duala</td>
</tr>
<tr>
<td>bambara</td>
<td>dutch</td>
</tr>
<tr>
<td>basaa</td>
<td>dzongkha</td>
</tr>
<tr>
<td>basque</td>
<td>embu</td>
</tr>
<tr>
<td>belarusan</td>
<td>english-au</td>
</tr>
<tr>
<td>bemba</td>
<td>english-australia</td>
</tr>
<tr>
<td>bena</td>
<td>english-ca</td>
</tr>
<tr>
<td>bengali</td>
<td>english-canada</td>
</tr>
<tr>
<td>bodo</td>
<td>english-gb</td>
</tr>
<tr>
<td>bosnian-cyrillic</td>
<td>english-newzealand</td>
</tr>
<tr>
<td>bosnian-cyrl</td>
<td>english-nz</td>
</tr>
<tr>
<td>bosnian-latin</td>
<td>english-unitedkingdom</td>
</tr>
<tr>
<td>bosnian-latin</td>
<td>english-unitedstates</td>
</tr>
<tr>
<td>bosnian</td>
<td>english-us</td>
</tr>
<tr>
<td>brazilian</td>
<td>english</td>
</tr>
<tr>
<td>breton</td>
<td>esperanto</td>
</tr>
<tr>
<td>british</td>
<td>estonian</td>
</tr>
<tr>
<td>bulgarian</td>
<td>ewe</td>
</tr>
<tr>
<td>burmese</td>
<td>ewondo</td>
</tr>
<tr>
<td>canadian</td>
<td>faroese</td>
</tr>
<tr>
<td>cantonese</td>
<td>filipino</td>
</tr>
<tr>
<td>catalan</td>
<td>finnish</td>
</tr>
<tr>
<td>centralatlastamazight</td>
<td>french-be</td>
</tr>
<tr>
<td>centralalkurdish</td>
<td>french-belgium</td>
</tr>
<tr>
<td>chechen</td>
<td>french-ca</td>
</tr>
<tr>
<td>cherokee</td>
<td>french-canada</td>
</tr>
<tr>
<td>chiga</td>
<td>french-ch</td>
</tr>
<tr>
<td>chinese-hans-hk</td>
<td>french-lu</td>
</tr>
</tbody>
</table>

12The name in the CLDR is Old Church Slavonic Cyrillic, but it has been shortened for practical reasons.
Modifying and adding values to ini files

New 3.39 There is a way to modify the values of ini files when they get loaded with
\babelprovide and import. To set, say, digits.native in the numbers section, use something like numbers/digits.native=abcdefghij. Keys may be added, too. Without import you may modify the identification keys. This can be used to create private variants easily. All you need is to import the same ini file with a different locale name and different parameters.

1.14 Selecting fonts

\babelfont[]\langle language-list\rangle\{⟨font-family\rangle\}[⟨font-options\rangle]{⟨font-name\rangle}

NOTE See the note in the previous section about some issues in specific languages.

The main purpose of \babelfont is to define at once in a multilingual document the fonts required by the different languages, with their corresponding language systems (script and language). So, if you load, say, 4 languages, \babelfont{rm}{FreeSerif} defines 4 fonts (with their variants, of course), which are switched with the language by babel. It is a tool to make things easier and transparent to the user. Here font-family is rm, sf or tt (or newly defined ones, as explained below), and font-name is the same as in fontspec and the like.

If no language is given, then it is considered the default font for the family, activated when a language is selected.

On the other hand, if there is one or more languages in the optional argument, the font will be assigned to them, overriding the default one. Alternatively, you may set a font for a script – just precede its name (lowercase) with a star (eg. *devanagari). With this optional argument, the font is not yet defined, but just predeclared. This means you may define as many fonts as you want ‘just in case’, because if the language is never selected, the corresponding \babelfont declaration is just ignored.

Babel takes care of the font language and the font script when languages are selected (as well as the writing direction); see the recognized languages above. In most cases, you will not need font-options, which is the same as in fontspec, but you may add further key/value pairs if necessary.

EXAMPLE Usage in most cases is very simple. Let us assume you are setting up a document in Swedish, with some words in Hebrew, with a font suited for both languages.

\documentclass{article}
\usepackage[swedish, bidi=default]{babel}
\babelprovide{import}{hebrew}
\babelfont{rm}{FreeSerif}
\begin{document}
Svenska \foreignlanguage{hebrew}{תיִרְבִע svenska.}
\end{document}

If on the other hand you have to resort to different fonts, you can replace the red line above with, say:

\babelfont{rm}{Iwona}
\babelfont[hebrew]{rm}{FreeSerif}

\[\textsuperscript{13}\textit{See also the package combofont for a complementary approach.}\]
\babelfont can be used to implicitly define a new font family. Just write its name instead of \texttt{rm}, \texttt{sf} or \texttt{tt}. This is the preferred way to select fonts in addition to the three basic families.

**EXAMPLE** Here is how to do it:

\begin{lstlisting}[language=luatex/xetex] \babelfont{kai}{FandolKai} \end{lstlisting}

Now, \texttt{kai} family and \texttt{kaifamily}, as well as \texttt{textkai} are at your disposal.

**NOTE** You may load fonts explicitly. For example:

\begin{lstlisting}[language=luatex/xetex] \usepackage{fontspec} \newfontscript{Devanagari}{deva} \babelfont[hindi]{rm}{Shobhika} \end{lstlisting}

This makes sure the OpenType script for Devanagari is deva and not dev2, in case it is not detected correctly. You may also pass some options to fontspec with \texttt{silent}, the warnings about unavailable scripts or languages are not shown (they are only really useful when the document format is being set up).

**NOTE** Directionality is a property affecting margins, indentation, column order, etc., not just text. Therefore, it is under the direct control of the language, which applies both the script and the direction to the text. As a consequence, there is no need to set \texttt{Script} when declaring a font with \babelfont (nor \texttt{Language}). In fact, it is even discouraged.

**NOTE** \texttt{fontspec} is not touched at all, only the preset font families (\texttt{rm}, \texttt{sf}, \texttt{tt}, and the like). If a language is switched when an \texttt{ad hoc} font is active, or you select the font with this command, neither the script nor the language is passed. You must add them by hand. This is by design, for several reasons—for example, each font has its own set of features and a generic setting for several of them can be problematic, and also preserving a “lower-level” font selection is useful.

**NOTE** The keys \texttt{Language} and \texttt{Script} just pass these values to the \texttt{font}, and do not set the script for the \texttt{language} (and therefore the writing direction). In other words, the ini file or \babelprovide provides default values for \babelfont if omitted, but the opposite is not true. See the note above for the reasons of this behavior.

**WARNING** Using \texttt{\setxxxxfont} and \babelfont at the same time is discouraged, but very often works as expected. However, be aware with \texttt{\setxxxxfont} the language system will not be set by babel and should be set with \texttt{fontspec} if necessary.

**TROUBLESHOOTING** Package fontspec Warning: ‘Language ‘LANG’ not available for font ‘FONT’ with script ‘SCRIPT’ ‘Default’ language used instead’. This is not an error. This warning is shown by fontspec, not by babel. It can be irrelevant for English, but not for many other languages, including Urdu and Turkish. This is a useful and harmless warning, and if everything is fine with your document the best thing you can do is just to ignore it altogether.

**TROUBLESHOOTING** Package babel Info: The following fonts are not babel standard families. This is not an error. babel assumes that if you are using \babelfont for a family, very likely you want to define the rest of them. If you don’t, you can find some inconsistencies between families. This checking is done at the beginning of the document, at a point where we cannot know which families will be used. Actually, there is no real need to use \babelfont in a monolingual document, if you set the language system in \texttt{\setmainfont} (or not, depending on what you want).

As the message explains, there is nothing intrinsically wrong with not defining all the families. In fact, there is nothing intrinsically wrong with not using \babelfont at all. But you must be aware that this may lead to some problems.

**NOTE** \babelfont is a high level interface to fontspec, and therefore in xetex you can apply mappings. For example, there is a set of **transliterations for Brahmic scripts** by Davis M. Jones. After installing them in your distribution, just set the map as you would do with fontspec.
1.15 Modifying a language

Modifying the behavior of a language (say, the chapter “caption”), is sometimes necessary, but not always trivial. In the case of caption names a specific macro is provided, because this is perhaps the most frequent change:

\setlocaption {⟨language-name⟩}{⟨caption-name⟩}{⟨string⟩}

New 3.51 Here caption-name is the name as string without the trailing name. An example, which also shows caption names are often a stylistic choice, is:

\setlocaption{english}{contents}{Table of Contents}

This works not only with existing caption names, because it also serves to define new ones by setting the caption-name to the name of your choice (name will be postpended). Captions so defined or redefined behave with the ‘new way’ described in the following note.

NOTE There are a few alternative methods:

• With data import’ed from ini files, you can modify the values of specific keys, like:

\babelprovide[import, captions/listtable = Lista de tablas]{spanish}

(In this particular case, instead of the captions group you may need to modify the captions.ltx one.)

• The ‘old way’, still valid for many languages, to redefine a caption is the following:

\addtocaptionsenglish{%
  \renewcommand\contentsname{Foo}%
}

As of 3.15, there is no need to hide spaces with % (babel removes them), but it is advisable to do so. This redefinition is not activated until the language is selected.

• The ‘new way’, which is found in bulgarian, azerbaijani, spanish, french, turkish, icelandic, vietnamese and a few more, as well as in languages created with \babelprovide and its key import, is:

\renewcommand\spanishchaptername{Foo}

This redefinition is immediate.

NOTE Do not redefine a caption in the following way:

\AtBeginDocument{\renewcommand\contentsname{Foo}}

The changes may be discarded with a language selector, and the original value restored.

Macros to be run when a language is selected can be add to \extras{lang}:

\addto\extrasrussian\{mymacro\}

There is a counterpart for code to be run when a language is unselected: \noextras{lang}.

NOTE These macros (\captions{lang}, \extras{lang}) may be redefined, but must not be used as such—they just pass information to babel, which executes them in the proper context.

Another way to modify a language loaded as a package or class option is by means of \babelprovide, described below in depth. So, something like:
first loads danish.ldf, and then redefines the captions for danish (as provided by the ini file) and prevents hyphenation. The rest of the language definitions are not touched. Without the optional argument it just loads some additional tools if provided by the ini file, like extra counters.

### 1.16 Creating a language

New 3.10 And what if there is no style for your language or none fits your needs? You may then define a quickly a language with the help of the following macro in the preamble (which may be used to modify an existing language, too, as explained in the previous subsection).

\begin{verbatim}
\babelprovide[\langle options \rangle]{\langle language-name \rangle}
\end{verbatim}

If the language \langle language-name \rangle has not been loaded as class or package option and there are no \langle options \rangle, it creates an “empty” one with some defaults in its internal structure: the hyphen rules, if not available, are set to the current ones, left and right hyphen mins are set to 2 and 3. In either case, caption, date and language system are not defined.

If no ini file is imported with import, \langle language-name \rangle is still relevant because in such a case the hyphenation and like breaking rules (including those for South East Asian and CJK) are based on it as provided in the ini file corresponding to that name; the same applies to OpenType language and script.

Conveniently, some options allow to fill the language, and babel warns you about what to do if there is a missing string. Very likely you will find alerts like that in the log file:

\begin{verbatim}
Package babel Warning: \chaptername not set for 'mylang'. Please,
\end{verbatim}

(babel) define it after the language has been loaded

(babel) (typically in the preamble) with:

(babel) \setlocalecaption{mylang}{chapter}{..}

(babel) Reported on input line 26.

In most cases, you will only need to define a few macros. Note languages loaded on the fly are not yet available in the preamble.

**EXAMPLE** If you need a language named arhinish:

\begin{verbatim}
\usepackage[danish]{babel}
\babelprovide{arhinish}
\setlocalecaption{arhinish}{chapter}{Chapitula}
\setlocalecaption{arhinish}{refname}{Refirenke}
\renewcommand{arhinishhyphenmins}{22}
\end{verbatim}

**EXAMPLE** Locales with names based on BCP 47 codes can be created with something like:

\begin{verbatim}
\babelprovide[import=en-US]{enUS}
\end{verbatim}

Note, however, mixing ways to identify locales can lead to problems. For example, is yi the name of the language spoken by the Yi people or is it the code for Yiddish?

The main language is not changed (danish in this example). So, you must add

\begin{verbatim}
\selectlanguage{arhinish}
\end{verbatim}

or other selectors where necessary.

If the language has been loaded as an argument in \documentclass or \usepackage, then \babelprovide redefines the requested data.
import= \langle language-tag \rangle

New 3.13 Imports data from an ini file, including captions and date (also line breaking rules in newly defined languages). For example:

\babelprovide[import=hu]{hungarian}

Unicode engines load the UTF-8 variants, while 8-bit engines load the LICR (ie, with macros like \' or \ss) ones.

New 3.23 It may be used without a value. In such a case, the ini file set in the corresponding babel-<language>.tex (where <language> is the last argument in \babelprovide) is imported. See the list of recognized languages above. So, the previous example can be written:

\babelprovide[import]{hungarian}

There are about 250 ini files, with data taken from the ldf files and the CLDR provided by Unicode. Not all languages in the latter are complete, and therefore neither are the ini files. A few languages may show a warning about the current lack of suitability of some features.

Besides \today, this option defines an additional command for dates: \<language>date, which takes three arguments, namely, year, month and day numbers. In fact, \today calls \<language>today, which in turn calls \<language>date\{\the\year\}\{\the\month\}\{\the\day\}. New 3.44 More convenient is usually \localedate, with prints the date for the current locale.

captions= \langle language-tag \rangle

Loads only the strings. For example:

\babelprovide[captions=hu]{hungarian}

hyphenrules= \langle language-list \rangle

With this option, with a space-separated list of hyphenation rules, babel assigns to the language the first valid hyphenation rules in the list. For example:

\babelprovide[hyphenrules=chavacano spanish italian]{chavacano}

If none of the listed hyphen rules exist, the default behavior applies. Note in this example we set chavacano as first option – without it, it would select spanish even if chavacano exists.

A special value is +, which allocates a new language (in the \TeX sense). It only makes sense as the last value (or the only one; the subsequent ones are silently ignored). It is mostly useful with \luatex, because you can add some patterns with \belpatterns, as for example:

\babelprovide[hyphenrules=+]{neo}
\belpatterns[neo]{a1 e1 i1 o1 u1}

In other engines it just suppresses hyphenation (because the pattern list is empty).

New 3.58 Another special value is unhyphenated, which activates a line breaking mode that allows spaces to be stretched to arbitrary amounts.
main This valueless option makes the language the main one (thus overriding that set when babel is loaded). Only in newly defined languages.

**EXAMPLE** Let’s assume your document (xetex or luatex) is mainly in Polytonic Greek with but with some sections in Italian. Then, the first attempt should be:

```latex
\usepackage[italian, greek.polutonic]{babel}
```

But if, say, accents in Greek are not shown correctly, you can try

```latex
\usepackage[italian, polytonicgreek, provide=*]{babel}
```

Remember there is an alternative syntax for the latter:

```latex
\usepackage[italian]{babel}
\babelprovide[import, main]{polytonicgreek}
```

Finally, also remember you might not need to load italian at all if there are only a few word in this language (see 1.3).

**script** *(script-name)*

New 3.15 Sets the script name to be used by fontspec (eg, Devanagari). Overrides the value in the ini file. If fontspec does not define it, then babel sets its tag to that provided by the ini file. This value is particularly important because it sets the writing direction, so you must use it if for some reason the default value is wrong.

**language** *(language-name)*

New 3.15 Sets the language name to be used by fontspec (eg, Hindi). Overrides the value in the ini file. If fontspec does not define it, then babel sets its tag to that provided by the ini file. Not so important, but sometimes still relevant.

**alph** *(counter-name)*

Assigns to \alph that counter. See the next section.

**Alph** *(counter-name)*

Same for \Alph.

A few options (only luatex) set some properties of the writing system used by the language. These properties are always applied to the script, no matter which language is active. Although somewhat inconsistent, this makes setting a language up easier in most typical cases.

**onchar** ids | fonts

New 3.38 This option is much like an ‘event’ called when a character belonging to the script of this locale is found (as its name implies, it acts on characters, not on spaces). There are currently two ‘actions’, which can be used at the same time (separated by a space): with ids the \language and the \localeid are set to the values of this locale; with fonts, the fonts are changed to those of this locale (as set with \babelfont). This option is not compatible with \mapfont. Characters can be added or modified with \babelfontproperty.

**NOTE** An alternative approach with luatex and Harfbuzz is the font option

RawFeature={multiscript=auto}. It does not switch the babel language and therefore the line breaking rules, but in many cases it can be enough.
intraspacesetstheinterwordspaceforthewritingsystemofthelanguagewithinemunits(so,0.10emplus0.1em).likespaceskip,themeunitappliedisthatofthecurrenttext(moreprecisely,thepreviousglyph).currentlyusedonlyinSoutheastAsianscripts,likethai,andCJK.

intrapenaltysetstheinterwordpenaltyforthewritingsystemofthislanguage.currentlyusedonlyinSoutheastAsianscripts,likeThai.ignoredif0(whichthisdefaultvalue).

transforms

Seesection1.21.

kashida|elongated|unhyphenated

New3.59Thererecurrentlythreeoptions,mainlyfortheArabicscript.Itsetsthe
linebreakingandjustificationmethod,whichcanbebasedonthethetraditionaltatweeling
characterorinthe'justificationalternatives'OpenTypetable(jalt).Foranexplanation
seethebabelsite.

linebreaking

New3.59Justasynonymousforjustification.

mapfont

Assignsthefontforthewritingdirectionofthislanguage(onlywithbidi=basic).
Wheneverpossible,insteadofthisoptionuseonchar,basedonthescript,whichusually
makesthesensemore.Moreprecisely,whatmapfont=directionmeansis,'whenacharacter
hasthesam dire ctionasthescriptforthe"provided"languagethent changer itsfontto
thatsetforthislanguage'.Thereare3directions,followingthebidiUnicodealgorithm,
namely,Arabic-lite,Hebrew-liteandlefttoright.Sothere should be atmost3directives
ofthistype.

NOTE(1)Ifyouneedshorthands,youcandefinedthemwith\useshorthandsand\defineshorthand
asdescribedabove.(2)Captionsand\todayare"ensured"with\babelensure(thisisthedefault
inini-basedlanguages).

1.17Digitsandcounters

New3.20Aboutthirtyinifilesdefin Feldnameddigits.native.Whenitispresent,
twomacrosarecreated:<language>digitsand<language>counteronlyxetexand
luatex).Withthefirst,astringof‘Latin’digitsareconvertedtothenativedigitsofthat
language;thesecondtakesacounternameasargument.Withtheoptionmaparabicin
\babelprovide,\arabicistheredefinedtoproduce.thenativedigits(thisisdoneglobally,tor
avoidinconsistenciesin,forexample,pagenumbering,andnoteaswelldatesdonotrely
on\arabic.).

Forexample:

\babelprovide[import]{telugu}
%Oralso,ifyouwant:
%\babelprovide[import,maparabic]{telugu}
\babelfont{rm}{Gautami}Withluatex,betterwithHarfbuzz
\begin{document}
\telugudigits{1234}
\telugucounter{section}
\end{document}

Languagesprovidingnativedigitsinallorsomevariantsare:
With luatex there is an alternative approach for mapping digits, namely, `mapdigits`. Conversion is based on the language and it is applied to the typeset text (not math, PDF bookmarks, etc.) before bidi and fonts are processed (i.e., to the node list as generated by the TeX code). This means the local digits have the correct bidirectional behavior (unlike `Numbers=Arabic` in fontspec, which is not recommended).

**NOTE** With xetex you can use the option `Mapping` when defining a font.

Many 'ini' locale files have been extended with information about non-positional numerical systems, based on those predefined in CSS. They only work with xetex and luatex and are fully expendable (even inside an unprotected \edef). Currently, they are limited to numbers below 10000.

There are several ways to use them (for the available styles in each language, see the list below):

- \localenumeral{⟨style⟩}{⟨number⟩}, like \localenumeral{abjad}{15}
- \localecounter{⟨style⟩}{⟨counter⟩}, like \localecounter{lower}{section}
- In \babelprovide, as an argument to the keys \alph and \Alph, which redefine what \alpha and \Alpha print. For example:

\begin{verbatim}
\babelprovide[alph=alphabetic]{thai}
\end{verbatim}

The styles are:

**Ancient Greek** lower.ancient, upper.ancient
**Amharic** afar, agaw, ari, blin, dizi, gedeo, gumuz, hadiyya, harari, kaffa, kebena, kembata, konso, kunama, meen, oromo, saho, sidama, silti, tigre, wolaita, yems
**Arabic** abjad, maghrebi.abjad
**Belarusan, Bulgarian, Macedonian, Serbian** lower, upper
**Bengali** alphabetic
**Coptic** epact,lower.letters
**Hebrew** letters (neither gershf nor gershayim yet)
**Hindi** alphabetic
**Armenian** lower.letter, upper.letter
**Japanese** hiragana, hiragana.iroha, katakana, katakana.iroha, circled.katakana, informal, formal, cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha
**Georgian** letters
**Greek** lower.modern, upper.modern, lower.ancient, upper.ancient (all with keraia)
**Khmer** consonant
**Korean** consonant, syllabe, hanja.informal, hanja.formal, hangul.formal, cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha
**Marathi** alphabetic
**Persian** abjad, alphabetic
**Russian** lower, lower.full, upper, upper.full
**Syriac** letters
**Tamil** ancient
**Thai** alphabetic
New 3.45 In addition, native digits (in languages defining them) may be printed with the numeral style digits.

1.18 Dates

New 3.45 When the data is taken from an ini file, you may print the date corresponding to the Gregorian calendar and other lunisolar systems with the following command.

\localedate \{\langle calendar=.., variant=..\rangle\}\{\langle year\rangle\}\{\langle month\rangle\}\{\langle day\rangle\}

By default the calendar is the Gregorian, but an ini file may define strings for other calendars (currently ar, ar-*, he, fa, hi). In the latter case, the three arguments are the year, the month, and the day in those in the corresponding calendar. They are not the Gregorian data to be converted (which means, say, 13 is a valid month number with calendar=hebrew).

Even with a certain calendar there may be variants. In Kurmanji the default variant prints something like 30. Çileya Pêşîn 2019, but with variant=izafa it prints 31’ê Çileya Pêşînê 2019.

1.19 Accessing language info

\languagename The control sequence \languagename contains the name of the current language.

WARNING Due to some internal inconsistencies in catcodes, it should not be used to test its value.

Use iflang, by Heiko Oberdiek.

\iflanguage \{\langle language\rangle\}\{\langle true\rangle\}\{\langle false\rangle\}

If more than one language is used, it might be necessary to know which language is active at a specific time. This can be checked by a call to \iflanguage, but note here “language” is used in the \TeX sense, as a set of hyphenation patterns, and not as its babel name. This macro takes three arguments. The first argument is the name of a language; the second and third arguments are the actions to take if the result of the test is true or false respectively.

\localeinfo \{\langle field\rangle\}

New 3.38 If an ini file has been loaded for the current language, you may access the information stored in it. This macro is fully expandable, and the available fields are:

name.english as provided by the Unicode CLDR.
tag.ini is the tag of the ini file (the way this file is identified in its name).
tag.bcp47 is the full BCP 47 tag (see the warning below).
language.tag.bcp47 is the BCP 47 language tag.
tag.openotype is the tag used by OpenType (usually, but not always, the same as BCP 47).
script.name, as provided by the Unicode CLDR.
script.tag.bcp47 is the BCP 47 tag of the script used by this locale.
script.tag.openotype is the tag used by OpenType (usually, but not always, the same as BCP 47).

WARNING New 3.46 As of version 3.46 tag.bcp47 returns the full BCP 47 tag. Formerly it returned just the language subtag, which was clearly counterintuitive.
\getlocaleproperty \{macro\}\{locale\}\{property\}

New 3.42 The value of any locale property as set by the ini files (or added/modified with \ babelprovide) can be retrieved and stored in a macro with this command. For example, after:

\getlocaleproperty\hechap{hebrew}\{captions/chapter\}

the macro \hechap will contain the string קרפ.
If the key does not exist, the macro is set to \relax and an error is raised. New 3.47 With the starred version no error is raised, so that you can take your own actions with undefined properties.
Babel remembers which ini files have been loaded. There is a loop named \LocaleForEach to traverse the list, where \#1 is the name of the current item, so that \LocaleForEach{\message{ **#1** }} just shows the loaded ini's.

NOTE ini files are loaded with \babelprovide and also when languages are selected if there is a \babelfont. To ensure the ini files are loaded (and therefore the corresponding data) even if these two conditions are not met, write \BabelEnsureInfo in the preamble.

\localeid

Each language in the babel sense has its own unique numeric identifier, which can be retrieved with \localeid.

NOTE The \localeid is not the same as the \language identifier, which refers to a set of hyphenation patterns (which, in turn, is just a component of the line breaking algorithm described in the next section). The data about preloaded patterns are store in an internal macro named \bbl@languages (see the code for further details), but note several locales may share a single \language, so they are separated concepts. In luatex, the \localeid is saved in each node (where it makes sense) as an attribute, too.

1.20 Hyphenation and line breaking

Babel deals with three kinds of line breaking rules: Western, typically the LGC group, South East Asian, like Thai, and CJK, but support depends on the engine: pdftex only deals with the former; xetex also with the second one (although in a limited way), while luatex provides basic rules for the latter, too. With luatex there are also tools for non-standard hyphenation rules, explained in the next section.

\babelhyphen\{type\}

\babelhyphen\{text\}

New 3.9a It is customary to classify hyphens in two types: (1) explicit or hard hyphens, which in \TeX are entered as -, and (2) optional or soft hyphens, which are entered as \-. Strictly, a soft hyphen is not a hyphen, but just a breaking opportunity or, in \TeX terms, a “discretionary”; a hard hyphen is a hyphen with a breaking opportunity after it. A further type is a non-breaking hyphen, a hyphen without a breaking opportunity.
In \TeX, - and \- forbid further breaking opportunities in the word. This is the desired behavior very often, but not always, and therefore many languages provide shorthands for these cases. Unfortunately, this has not been done consistently: for example, " - in Dutch, Portuguese, Catalan or Danish is a hard hyphen, while in German, Spanish, Norwegian, Slovak or Russian is a soft hyphen. Furthermore, some of them even redefine \-, so that you cannot insert a soft hyphen without breaking opportunities in the rest of the word. Therefore, some macros are provided with a set of basic “hyphens” which can be used by themselves, to define a user shorthand, or even in language files.

- \babelhyphen\{soft\} and \babelhyphen\{hard\} are self explanatory.
- \babelhyphen\{repeat\} inserts a hard hyphen which is repeated at the beginning of the next line, as done in languages like Polish, Portuguese and Spanish.
• \textbackslash{babelhyphen}{nobreak} inserts a hard hyphen without a break after it (even if a space follows).

• \textbackslash{babelhyphen}{empty} inserts a break opportunity without a hyphen at all.

• \textbackslash{babelhyphen}{(text)} is a hard “hyphen” using \textbackslash{text} instead. A typical case is \textbackslash{babelhyphen}/.

With all of them, hyphenation in the rest of the word is enabled. If you don’t want to enable it, there is a starred counterpart: \textbackslash{babelhyphen}*{soft} (which in most cases is equivalent to the original \textbackslash{-}), \textbackslash{babelhyphen}*{hard}, etc.

Note hard is also good for isolated prefixes (eg, \textbackslash{anti}) and nobreak for isolated suffixes (eg, \textbackslash{-ism}), but in both cases \textbackslash{babelhyphen}*{nobreak} is usually better.

There are also some differences with \textbackslash{LATEX}: (1) the character used is that set for the current font, while in \textbackslash{LATEX} it is hardwired to - (a typical value); (2) the hyphen to be used in fonts with a negative \textbackslash{hyphenchar} is -, like in \textbackslash{LATEX}, but it can be changed to another value by redefining \textbackslash{babelnul1hyphen}; (3) a break after the hyphen is forbidden if preceded by a glue >0 pt (at the beginning of a word, provided it is not immediately preceded by, say, a parenthesis).

\textbackslash{babelhyphenation} \{\langle language \rangle, \langle language \rangle, ...\}\{\langle exceptions \rangle\}

\textbf{New 3.9a} Sets hyphenation exceptions for the languages given or, without the optional argument, for all languages (eg, proper nouns or common loan words, and of course monolingual documents). Language exceptions take precedence over global ones.

It can be used only in the preamble, and exceptions are set when the language is first selected, thus taking into account changes of \textbackslash{lccodes}’s done in \textbackslash{extras}\{\langle lang \rangle\} as well as the language-specific encoding (not set in the preamble by default). Multiple \textbackslash{babelhyphenation’s} are allowed. For example:

\textbackslash{babelhyphenation}\{Wal-hal-la Dar-bhan-ga\}

Listed words are saved expanded and therefore it relies on the LICR. Of course, it also works without the LICR if the input and the font encodings are the same, like in Unicode based engines.

\textbf{NOTE} Using \textbackslash{babelhyphenation} with Southeast Asian scripts is mostly pointless. But with \textbackslash{babelpatterns} (below) you may fine-tune line breaking (only \textbackslash{luatex}). Even if there are no patterns for the language, you can add at least some typical cases.

\textbf{NOTE} To set hyphenation exceptions in the preamble before any language is explicitly set with a selector, use \textbackslash{babelhyphenation} instead of \textbackslash{hyphenation}. In the preamble the \textbackslash{hyphenation} rules are not always fully set up and an error can be raised.

\textbackslash{begin}\{hyphenrules\}\{\langle language \rangle\} ... \textbackslash{end}\{hyphenrules\}

The environment \textbackslash{hyphenrules} can be used to select only the hyphenation rules to be used (it can be used as command, too). This can for instance be used to select ‘nohyphenation’, provided that in \textbackslash{language}.dat the ‘language’ nohyphenation is defined by loading \textbackslash{zerohyph.tex}. It deactivates language shorthands, too (but not user shorthands).

Except for these simple uses, \textbackslash{hyphenrules} is deprecated and other \textbackslash{language}* (the starred version) is preferred, because the former does not take into account possible changes in encodings of characters like, say, ‘done by some languages (eg, italian, french, ukraineb).

\textbackslash{babelpatterns} \{\langle language \rangle, \langle language \rangle, ...\}\{\langle patterns \rangle\}

\textbf{New 3.9m} In \textbackslash{luatex} only,\textsuperscript{14} adds or replaces patterns for the languages given or, without the optional argument, for \textit{all} languages. If a pattern for a certain combination already exists, it gets replaced by the new one.

\textsuperscript{14}With \textbackslash{luatex} exceptions and patterns can be modified almost freely. However, this is very likely a task for a separate package and \texttt{babel} only provides the most basic tools.
It can be used only in the preamble, and patterns are added when the language is first selected, thus taking into account changes of \lccodes's done in \extras\langle\text{lang}\rangle as well as the language-specific encoding (not set in the preamble by default). Multiple \belpattern's are allowed. 

Listed patterns are saved expanded and therefore it relies on the LICR. Of course, it also works without the LICR if the input and the font encodings are the same, like in Unicode based engines.

**New 3.31** (Only \luatex.) With \belprovide and imported CJK languages, a simple generic line breaking algorithm (push-out-first) is applied, based on a selection of the Unicode rules (**New 3.32** it is disabled in verbatim mode, or more precisely when the hyphenrules are set to nohyphenation). It can be activated alternatively by setting explicitly the intraspace.

**New 3.27** Interword spacing for Thai, Lao and Khmer is activated automatically if a language with one of these scripts are loaded with \belprovide. See the sample on the babel repository. With both Unicode engines, spacing is based on the “current” em unit (the size of the previous char in \luatex, and the font size set by the last \selectfont in \xetex).

### 1.21 Transforms

Transforms (only \luatex) provide a way to process the text on the typesetting level in several language-dependent ways, like non-standard hyphenation, special line breaking rules, script to script conversion, spacing conventions and so on.\[^{15}\] It currently embraces \belprehyphenation and \belposthyphenation.

**New 3.57** Several ini files predefine some transforms. They are activated with the key \transforms in \belprovide, either if the locale is being defined with this macro or the languages has been previously loaded as a class or package option, as the following example illustrates:

```latex
\usepackage[magyar]{babel}
\belprovide[{transforms = digraphs.hyphen}]{magyar}
```

**New 3.67** Transforms predefined in the ini locale files can be made attribute-dependent, too. When an attribute between parenthesis is inserted subsequent transforms will be assigned to it (up to the list end or another attribute). For example, and provided an attribute called \withsigmafinal has been declared:

```
transforms = transliteration.omega (\withsigmafinal) sigma.final
```

This applies transliteration.omega always, but sigma.final only when \withsigmafinal is set.

Here are the transforms currently predefined. (More to follow in future releases.)

<table>
<thead>
<tr>
<th>Language</th>
<th>Transform</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>transliteration.dad</td>
<td>Applies the transliteration system devised by Yannis Haralambous for dad (simple and \TeX-friendly). Not yet complete, but sufficient for most texts.</td>
</tr>
<tr>
<td>Croatian</td>
<td>digraphs.ligatures</td>
<td>Ligatures $\text{DŽ}$, $\text{DŽ}$, $\text{LJ}$, $\text{LJ}$, $\text{Nj}$, $\text{Nj}$, $\text{nj}$. It assumes they exist. This is not the recommended way to make these transformations (the best way is with OTF features), but it can get you out of a hurry.</td>
</tr>
<tr>
<td>Czech, Polish, Portuguese, Slovak, Spanish</td>
<td>hyphen.repeat</td>
<td>Explicit hyphens behave like \belprehyphen {repeat}.</td>
</tr>
</tbody>
</table>

\[^{15}\]They are similar in concept, but not the same, as those in Unicode. The main inspiration for this feature is the Omega transformation processes.
<table>
<thead>
<tr>
<th>Language</th>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Czech, Polish, Slovak</td>
<td>oneletter.nobreak</td>
<td>Converts a space after a non-syllabic preposition or conjunction into a non-breaking space.</td>
</tr>
<tr>
<td>Finnish</td>
<td>prehyphen.nobreak</td>
<td>Line breaks just after hyphens prepended to words are prevented, like in “pakasteakaapit ja-arkut”.</td>
</tr>
<tr>
<td>Greek</td>
<td>diaeresis.hyphen</td>
<td>Removes the diaeresis above iota and upsilon if hyphenated just before. It works with the three variants.</td>
</tr>
<tr>
<td>Greek</td>
<td>transliteration.omega</td>
<td>Although the provided combinations are not the full set, this transform follows the syntax of Omega: ( = ) for the circumflex, ( \nu ) for digamma, and so on. For better compatibility with Levy’s system, ( ~ ) (as ‘string’) is an alternative to ( = ). ‘( \iota )’ is tons in Monotonic Greek, but oxia in Polytonic and Ancient Greek.</td>
</tr>
<tr>
<td>Greek</td>
<td>sigma.final</td>
<td>The transliteration system above does not convert the sigma at the end of a word (on purpose). This transforms does it. To prevent the conversion (an abbreviation, for example), write “s.”</td>
</tr>
<tr>
<td>Hindi, Sanskrit</td>
<td>transliteration.hk</td>
<td>The Harvard-Kyoto system to romanize Devanagari.</td>
</tr>
<tr>
<td>Hindi, Sanskrit</td>
<td>punctuation.space</td>
<td>Inserts a space before the following four characters: !?:; .</td>
</tr>
<tr>
<td>Hungarian</td>
<td>digraphs.hyphen</td>
<td>Hyphenates the long digraphs ccs, ddz, ggy, lly, nny, ssz, tty and zzs as cs-cs, dz-dz, etc.</td>
</tr>
<tr>
<td>Indic scripts</td>
<td>danda.nobreak</td>
<td>Prevents a line break before a danda or double danda if there is a space. For Assamese, Bengali, Gujarati, Hindi, Kannada, Malayalam, Marathi, Oriya, Tamil, Telugu.</td>
</tr>
<tr>
<td>Latin</td>
<td>digraphs.ligatures</td>
<td>Replaces the groups ae, AE, oe, OE with ( a ), ( A ), ( o ), ( O ).</td>
</tr>
<tr>
<td>Latin</td>
<td>letters.noj</td>
<td>Replaces j, J with i, I.</td>
</tr>
<tr>
<td>Latin</td>
<td>letters.uv</td>
<td>Replaces v, U with u, V.</td>
</tr>
<tr>
<td>Sanskrit</td>
<td>transliteration.iast</td>
<td>The IAST system to romanize Devanagari.¹⁶</td>
</tr>
<tr>
<td>Serbian</td>
<td>transliteration.gajica</td>
<td>(Note serbian with ini files refers to the Cyrillic script, which is here the target.) The standard system devised by Ljudevit Gaj.</td>
</tr>
<tr>
<td>Arabic, Persian</td>
<td>kashida.plain</td>
<td>Experimental. A very simple and basic transform for ‘plain’ Arabic fonts, which attempts to distribute the tatwil as evenly as possible (starting at the end of the line). See the news for version 3.59.</td>
</tr>
</tbody>
</table>

\( \texttt{\backslash babelposthyphenation} \{\langle \text{options} \rangle\}\{\langle \text{hyphenrules-name} \rangle\}\{\langle \text{lua-pattern} \rangle\}\{\langle \text{replacement} \rangle\}\) \( \backslash \) \texttt{babelposthyphenation} 

**New 3.37-3.39** With \texttt{luatex} it is possible to define non-standard hyphenation rules, like \( f-f \rightarrow ff-f \), repeated hyphens, ranked ruled (or more precisely, ‘penalized’ hyphenation points), and so on. A few rules are currently provided (see above), but they can be defined as shown in the following example, where \{1\} is the first captured char (between ( ) in the pattern):
In the replacements, a captured char may be mapped to another, too. For example, if the first capture reads \([\i\u]\), the replacement could be \{1|\i|\u\}, which maps \(i\) to \(i\), and \(\u\) to \(\u\), so that the diaeresis is removed.

This feature is activated with the first \texttt{\babelposthyphenation} or \texttt{\babelprehyphenation}.

\begin{verbatim}
\babelposthyphenation{german}{\{fmtrp\}} {1}
{
  { no = {1}, pre = {1}{1}- }, % Replace first char with disc
  remove, % Remove automatic disc (2nd node)
  } % Keep last char, untouched
\}
\end{verbatim}

\textbf{New 3.67} With the optional argument you can associate a user defined transform to an attribute, so that it's active only when it's set (currently its attribute value is ignored). With this mechanism transforms can be set or unset even in the middle of paragraphs, and applied to single words. To define, set and unset the attribute, the LaTeX kernel provides the macros \texttt{\newattribute}, \texttt{\setattribute} and \texttt{\unsetattribute}. The following example shows how to use it, provided an attribute named \texttt{\latinnoj} has been declared:

\begin{verbatim}
\babelprehyphenation[attribute=\latinnoj]{latin}{ J }{ string = I }
\end{verbatim}

See the \texttt{babel} site for a more detailed description and some examples. It also describes a few additional replacement types (\texttt{string}, \texttt{penalty}).

Although the main purpose of this command is non-standard hyphenation, it may actually be used for other transformations (after hyphenation is applied, so you must take discretionaries into account).

You are limited to substitutions as done by lua, although a future implementation may alternatively accept lpeg.

\texttt{\babelposthyphenation} \[(\text{options})\]{\langle\text{locale-name}\rangle}{\langle\text{lua-pattern}\rangle}{\langle\text{replacement}\rangle}

\textbf{New 3.44-3-52} It is similar to the latter, but (as its name implies) applied before hyphenation, which is particularly useful in transliterations. There are other differences:

1. the first argument is the locale instead of the name of the hyphenation patterns;
2. in the search patterns = has no special meaning, while \texttt{|} stands for an ordinary space;
3. in the replacement, discretionaries are not accepted.

See the description above for the optional argument. This feature is activated with the first \texttt{\babelposthyphenation} or \texttt{\babelprehyphenation}.

\textbf{EXAMPLE} You can replace a character (or series of them) by another character (or series of them).

Thus, to enter \(\acute{z}\) as \(zh\) and \(\check{s}\) as \(sh\) in a newly created locale for transliterated Russian:

\begin{verbatim}
\babelprovide[hyphenrules=+]{russian-latin} % Create locale
\babelprehyphenation{russian-latin}{([sz]h)} % Create rule
{
  string = {1|sz|$\check{s}$},
  remove
}
\end{verbatim}

\textbf{EXAMPLE} The following rule prevent the word “a” from being at the end of a line:

\begin{verbatim}
\babelprehyphenation{english}{|a|}
{
  {}, {}, % Keep first space and a
  { insert, penalty = 10000 }, % Insert penalty
  {} % Keep last space
}
\end{verbatim}
NOTE With \texttt{lualatex} there is another approach to make text transformations, with the function `\texttt{\textbackslash fonts\_handlers\_otf\_addfeature}`, which adds new features to an OTF font (substitution and positioning). These features can be made language-dependent, and babel by default recognizes this setting if the font has been declared with `\texttt{\textbackslash babelfont}`. The `transforms` mechanism supplements rather than replaces OTF features.

With \texttt{\textbackslash xetex}, where `\texttt{\textbackslash transforms}` are not available, there is still another approach, with font mappings, mainly meant to perform encoding conversions and transliterations. Mappings, however, are linked to fonts, not to languages.

### 1.22 Selection based on BCP 47 tags

**New 3.43** The recommended way to select languages is that described at the beginning of this document. However, BCP 47 tags are becoming customary, particularly in documents (or parts of documents) generated by external sources, and therefore babel will provide a set of tools to select the locales in different situations, adapted to the particular needs of each case. Currently, babel provides autoloading of locales as described in this section. In these contexts autoloading is particularly important because we may not know on beforehand which languages will be requested.

It must be activated explicitly, because it is primarily meant for special tasks. Mapping from BCP 47 codes to locale names are not hardcoded in babel. Instead the data is taken from the ini files, which means currently about 250 tags are already recognized. Babel performs a simple lookup in the following way: \texttt{fr-Latn-FR $\rightarrow$ fr-Latn $\rightarrow$ fr-FR $\rightarrow$ fr}.

Languages with the same resolved name are considered the same. Case is normalized before, so that \texttt{fr-latn-fr $\rightarrow$ fr-Latn-FR}. If a tag and a name overlap, the tag takes precedence.

Here is a minimal example:

```latex
\documentclass{article}
\usepackage[danish]{babel}
\babeladjust{
  autoload.bcp47 = on,
  autoload.bcp47.options = import
}
\begin{document}
\selectlanguage{de-AT}
\chapter{Chapter in Danish: \chaptername.}
\localedate{2020}{1}{30}
\end{document}
```

Currently the locales loaded are based on the ini files and decoupled from the main ldf files. This is by design, to ensure code generated externally produces the same result regardless of the languages requested in the document, but an option to use the ldf instead will be added in a future release, because both options make sense depending on the particular needs of each document (there will be some restrictions, however).

The behaviour is adjusted with `\texttt{\textbackslash babeladjust}` with the following parameters:

- `\texttt{autoload.bcp47 = on/off}`
- `\texttt{autoload.bcp47.options = import/none/ignore}`

You may add `\texttt{import}` (features defined in the corresponding babel-\ldots\texttt{.tex} file might not be available).
autoload.bcp47.prefix. Although the public name used in selectors is the tag, the internal name will be different and generated by prepending a prefix, which by default is bcp47-. You may change it with this key.

New 3.46 If an ldf file has been loaded, you can enable the corresponding language tags as selector names with:

\babeladjust{ bcp47.toname = on } (You can deactivate it with off.) So, if dutch is one of the package (or class) options, you can write \selectlanguage{nl}. Note the language name does not change (in this example is still dutch), but you can get it with \localeinfo or \getlanguageproperty. It must be turned on explicitly for similar reasons to those explained above.

1.23 Selecting scripts

Currently babel provides no standard interface to select scripts, because they are best selected with either \fontencoding (low-level) or a language name (high-level). Even the Latin script may require different encodings (ie, sets of glyphs) depending on the language, and therefore such a switch would be in a sense incomplete. Some languages sharing the same script define macros to switch it (eg, \textcyrillic), but be aware they may also set the language to a certain default. Even the babel core defined \textlatin, but is was somewhat buggy because in some cases it messed up encodings and fonts (for example, if the main Latin encoding was LY1), and therefore it has been deprecated.

{}\ensureascii\{⟨text⟩\} New 3.9i This macro makes sure ⟨text⟩ is typeset with a LICR-savvy encoding in the ASCII range. It is used to redefine \TeX and \LaTeX so that they are correctly typeset even with LGR or X2 (the complete list is stored in \BabelNonASCII, which by default is LGR, X2, OT2, OT3, OT6, LHE, LWN, LMA, LMC, LMS, LMU, but you can modify it). So, in some sense it fixes the bug described in the previous paragraph. If non-ASCII encodings are not loaded (or no encoding at all), it is no-op (also \TeX and \LaTeX are not redefined); otherwise, \ensureascii switches to the encoding at the beginning of the document if ASCII-savvy, or else the last ASCII-savvy encoding loaded. For example, if you load LY1, LGR, then it is set to LY1, but if you load LY1, T2A it is set to T2A. The symbol encodings TS1, T3, and T53 are not taken into account, since they are not used for “ordinary” text (they are stored in \BabelNonText, used in some special cases when no Latin encoding is explicitly set).

The foregoing rules (which are applied “at begin document”) cover most of the cases. No assumption is made on characters above 127, which may not follow the LICR conventions – the goal is just to ensure most of the ASCII letters and symbols are the right ones.

1.24 Selecting directions

No macros to select the writing direction are provided, either – writing direction is intrinsic to each script and therefore it is best set by the language (which can be a dummy one). Furthermore, there are in fact two right-to-left modes, depending on the language, which differ in the way ‘weak’ numeric characters are ordered (eg, Arabic %123 vs Hebrew 123%).

WARNING The current code for text in lualatex should be considered essentially stable, but, of course, it is not bug-free and there can be improvements in the future, because setting bidi text has many subtleties (see for example <https://www.w3.org/TR/html-bidi/>). A basic stable version for other

\footnote{The so-called Unicode fonts do not improve the situation either. So, a font suited for Vietnamese is not necessarily suited for, say, the romanization of Indic languages, and the fact it contains glyphs for Modern Greek does not mean it includes them for Classic Greek.}

\footnote{But still defined for backwards compatibility.}
engines must wait. This applies to text; there is a basic support for graphical elements, including the picture environment (with \texttt{pict2e}) and \texttt{pgf/tikz}. Also, indexes and the like are under study, as well as math (there are progresses in the latter, including \texttt{amsmath} and \texttt{mathtools} too, but for example gathered may fail).

An effort is being made to avoid incompatibilities in the future (this one of the reason currently bidi must be explicitly requested as a package option, with a certain bidi model, and also the layout options described below).

\textbf{WARNING} If characters to be mirrored are shown without changes with \texttt{luatex}, try with the following line:

\begin{verbatim}
\babeladjust{bidi.mirroring=off}
\end{verbatim}

There are some package options controlling bidi writing.

\begin{itemize}
\item \texttt{bidi=} \texttt{default} | \texttt{basic} | \texttt{basic-r} | \texttt{bidi-l} | \texttt{bidi-r}
\end{itemize}

\texttt{New 3.14} Selects the bidi algorithm to be used. With default the bidi mechanism is just activated (by default it is not), but every change must be marked up. In \texttt{xetex} and \texttt{pdftex} this is the only option. In \texttt{luatex}, \texttt{basic-r} provides a simple and fast method for \texttt{R} text, which handles numbers and unmarked \texttt{L} text within an \texttt{R} context many in typical cases. \texttt{New 3.19} Finally, basic supports both \texttt{L} and \texttt{R} text, and it is the preferred method (support for \texttt{basic-r} is currently limited). (They are named \texttt{basic} mainly because they only consider the intrinsic direction of scripts and weak directionality.)

\texttt{New 3.29} In \texttt{xetex}, \texttt{bidi-r} and \texttt{bidi-l} resort to the package \texttt{bidi} (by Vafa Khalighi). Integration is still somewhat tentative, but it mostly works. For \texttt{RL} documents use the former, and for \texttt{LR} ones use the latter. There are samples on GitHub, under /\texttt{required/babel/samples}. See particularly\texttt{lua-bidibasic.tex} and \texttt{lua-secenum.tex}.

\textbf{EXAMPLE} The following text comes from the Arabic Wikipedia (article about Arabia). Copy-pasting some text from the Wikipedia is a good way to test this feature. Remember \texttt{basic} is available in \texttt{luatex} only.

\begin{verbatim}
\documentclass{article}
\usepackage[bidi=basic]{babel}
\babelprovide{import, main}{arabic}
\babelfont{rm}{FreeSerif}
\begin{document}
ـﺑ(ﻲﻘﻳﺮﻏﻻا)ﻲﻨﻴﻠﻴﻬﻟاﺮﺼﻌﻟاﺔﻠﻴﻃبﺮﻌﻟاةﺮﻳﺰﺟﻪﺒﺷﺖﻓﺮﻋﺪﻗو
Arabia
وأ
Aravia )
ﺔﻴﻘﻳﺮﻏﻻا
(؟)ـﺑتﺎﺋدﺎﺑ
Arabia
أ
إ،

\end{document}
\end{verbatim}

\textbf{EXAMPLE} With \texttt{bidi=basic} both \texttt{L} and \texttt{R} text can be mixed without explicit markup (the latter will be only necessary in some special cases where the Unicode algorithm fails). It is used much like \texttt{bidi=\texttt{basic-r}}, but with \texttt{R} text inside \texttt{L} text you may want to map the font so that the correct features are in force. This is accomplished with an option in \texttt{\babelprovide}, as illustrated:

\begin{verbatim}
\documentclass{article}
\usepackage[bidi=\texttt{basic-r}]{babel}
\begin{document}
وبناددات

\end{document}
\end{verbatim}
Most Arabic speakers consider the two varieties to be two registers of one language, although the two registers can be referred to in Arabic as \textit{fuṣḥā l-ʻaṣr} (MSA) and \textit{fuṣḥā t-turāth} (CA).

In this example, and thanks to onchar=ids fonts, any Arabic letter (because the language is arabic) changes its font to that set for this language (here defined via *arabic, because Crimson does not provide Arabic letters).

NOTE Boxes are “black boxes”. Numbers inside an \hbox (for example in a \ref) do not know anything about the surrounding chars. So, \ref{A}-\ref{B} are not rendered in the visual order A-B, but in the wrong one B-A (because the hyphen does not “see” the digits inside the \hbox'es).

If you need \ref ranges, the best option is to define a dedicated macro like this (to avoid explicit direction changes in the body; here \texte must be defined to select the main language):

\newcommand\refrange[2]{\babelsublr{\texte{\ref{#1}}-\texte{\ref{#2}}}}

In the future a more complete method, reading recursively boxed text, may be added.

layout= sectioning | counters | lists | contents | footnotes | captions | columns | graphics | extras

New 3.16 To be expanded. Selects which layout elements are adapted in bidi documents, including some text elements (except with options loading the bidi package, which provides its own mechanism to control these elements). You may use several options with a dot-separated list (eg, layout=counters.contents.sectioning). This list will be expanded in future releases. Note not all options are required by all engines.

sectioning makes sure the sectioning macros are typeset in the main language, but with the title text in the current language (see below \BabelPatchSection for further details).

counters required in all engines (except luatex with bidi=basic) to reorder section numbers and the like (eg, \section); required in xetex and pdftex for counters in general, as well as in luatex with bidi=default; required in luatex for numeric footnote marks >9 with bidi=basic-r (but not with bidi=basic); note, however, it can depend on the counter format.

With counters, \arabic is not only considered L text always (with \babelsublr, see below), but also an “isolated” block which does not interact with the surrounding chars. So, while 1.2 in R text is rendered in that order with bidi=basic (as a decimal number), in \arabic(c1) \arabic(c2) the visual order is c2,c1. Of course, you may always adjust the order by changing the language, if necessary.\footnote{Next on the roadmap are counters and numeral systems in general. Expect some minor readjustments.}

lists required in xetex and pdftex, but only in bidirectional (with both R and L paragraphs) documents in luatex.
**WARNING** As of April 2019 there is a bug with `\parshape` in `luatex` (a TeX primitive) which makes lists to be horizontally misplaced if they are inside a `\vbox` (like `minipage`) and the current direction is different from the main one. A workaround is to restore the main language before the box and then set the local one inside.

**contents** required in `xetex` and `pdftex`; in `luatex` toc entries are `R` by default if the main language is `R`.

**columns** required in `xetex` and `pdftex` to reverse the column order (currently only the standard two-column mode); in `luatex` they are `R` by default if the main language is `R` (including `multicol`).

**footnotes** not required in monolingual documents, but it may be useful in bidirectional documents (with both `R` and `L` paragraphs) in all engines; you may use alternatively `\BabelFootnote` described below (what this option does exactly is also explained there).

**captions** is similar to sectioning, but for `\caption`; not required in monolingual documents with `luatex`, but may be required in `xetex` and `pdftex` in some styles (support for the latter two engines is still experimental) New 3.18.

**tabular** required in `luatex` for `R` `tabular`, so that the first column is the right one (it has been tested only with simple tables, so expect some readjustments in the future); ignored in `pdftex` or `xetex` (which will not support a similar option in the short term). It patches an internal command, so it might be ignored by some packages and classes (or even raise an error) New 3.18.

**graphics** modifies the `picture` environment so that the whole figure is `L` but the text is `R`. It does not work with the standard `picture`, and `pict2e` is required. It attempts to do the same for `pgf/tikz`. Somewhat experimental New 3.32.

**extras** is used for miscellaneous readjustments which do not fit into the previous groups. Currently redefines in `luatex` `\underline` and `\LaTeX2e` New 3.19.

**EXAMPLE** Typically, in an Arabic document you would need:

```latex
\usepackage[bidi=basic,layout=counters.tabular]{babel}
```

\babelsubl {⟨lr-text⟩}

Digits in `pdftex` must be marked up explicitly (unlike `luatex` with `bidi=basic` or `bidi=basic-r` and, usually, `xetex`). This command is provided to set `{⟨lr-text⟩}` in `L` mode if necessary. It's intended for what Unicode calls weak characters, because words are best set with the corresponding language. For this reason, there is no `rl` counterpart. Any `\babelsubl` in `explicit L` mode is ignored. However, with `bidi=basic` and `implicit L`, it first returns to `R` and then switches to `explicit L`. To clarify this point, consider, in an `R` context:

```
\ \RTL A ltr text \thechapter{} and still ltr RTL B
```

There are three `R` blocks and two `L` blocks, and the order is `RTL B and still ltr 1 ltr text RTL A`. This is by design to provide the proper behavior in the most usual cases — but if you need to use `\ref` in an `L` text inside `R`, the `L` text must be marked up explicitly; for example:

```
\RTL A \foreignlanguage{english}{ltr text \thechapter{} and still ltr} RTL B
```

\BabelPatchSection {⟨section-name⟩}

Mainly for `bidi` text, but it can be useful in other cases. `\BabelPatchSection` and the corresponding option `layout=sectioning` takes a more logical approach (at least in many cases) because it applies the global language to the section format (including the
\chaptername in \chapter, while the section text is still the current language. The latter is passed to tocs and marks, too, and with sectioning in layout they both reset the “global” language to the main one, while the text uses the “local” language. With layout=sectioning all the standard sectioning commands are redefined (it also “isolates” the page number in heads, for a proper bidi behavior), but with this command you can set them individually if necessary (but note then tocs and marks are not touched).

\BabelFootnote \{\langle cmd\rangle\}\{\langle local-language\rangle\}\{\langle before\rangle\}\{\langle after\rangle\}

New 3.17 Something like:

\BabelFootnote{\parsfootnote}{\languagename}{}{}%

defines \parsfootnote so that \parsfootnote{note} is equivalent to:

\footnote{\langle foreign-language\{\languagename\}\{note\}\rangle}

but the footnote itself is typeset in the main language (to unify its direction). In addition, \parsfootnotetext is defined. The option footnotes just does the following:

\BabelFootnote{\footnote}{\languagename}{}{}% \BabelFootnote{\localfootnote}{\languagename}{}{}% \BabelFootnote{\mainfootnote}{}{}{}

(which also redefine \footnotetext and define \localfootnotetext and \mainfootnotetext). If the language argument is empty, then no language is selected inside the argument of the footnote. Note this command is available always in bidi documents, even without layout=footnotes.

**EXAMPLE** If you want to preserve directionality in footnotes and there are many footnotes entirely in English, you can define:

\BabelFootnote{\enfootnote}{english}{}{.}

It adds a period outside the English part, so that it is placed at the left in the last line. This means the dot the end of the footnote text should be omitted.

1.25 Language attributes

\languageattribute

This is a user-level command, to be used in the preamble of a document (after \usepackage[...]{babel}), that declares which attributes are to be used for a given language. It takes two arguments: the first is the name of the language; the second, a (list of) attribute(s) to be used. Attributes must be set in the preamble and only once – they cannot be turned on and off. The command checks whether the language is known in this document and whether the attribute(s) are known for this language. Very often, using a modifier in a package option is better. Several language definition files use their own methods to set options. For example, french uses \frenchsetup, magyar (1.5) uses \magyarOptions; modifiers provided by spanish have no attribute counterparts. Macros setting options are also used (eg, \ProsodicMarksOn in latin).

1.26 Hooks

New 3.9a A hook is a piece of code to be executed at certain events. Some hooks are predefined when luatex and xetex are used.
This is not the only way to inject code at those points. The events listed below can be used as a hook name in `\AddToHook` in the form `\AddToHook{\langle language-name\rangle/\langle event-name\rangle}` (with * it's applied to all languages), but there is a limitation, because the parameters passed with the babel mechanism are not allowed. The `\AddToHook` mechanism does not replace the current one in 'babel'. Its main advantage is you can reconfigure 'babel' even before loading it. See the example below.

\AddToHook{\langle lang\rangle}{\langle name\rangle}{\langle event\rangle}{\langle code\rangle}

The same name can be applied to several events. Hooks with a certain `{\langle name\rangle}` may be enabled and disabled for all defined events with `\EnableBabelHook{\langle name\rangle}`, `\DisableBabelHook{\langle name\rangle}`. Names containing the string `babel` are reserved (they are used, for example, by `\useshortands*` to add a hook for the event `afterextras`).

`\EnableBabelHook` may also be applied to a specific language with the optional argument; language-specific settings are executed after global ones.

Current events are the following; in some of them you can use one to three `\TeX` parameters (#1, #2, #3), with the meaning given:

- `addialect` (language name, dialect name) Used by `luababel.def` to load the patterns if not preloaded.
- `patterns` (language name, language with encoding) Executed just after the `\language` has been set. The second argument has the patterns name actually selected (in the form of either `lang:ENC` or `lang`).
- `hyphenation` (language name, language with encoding) Executed locally just before exceptions given in `\babelhyphenation` are actually set.
- `defaultcommands` Used (locally) in `\StartBabelCommands`.
- `encodedcommands` (input, font encodings) Used (locally) in `\StartBabelCommands`. Both `xetex` and `luatex` make sure the encoded text is read correctly.
- `stopcommands` Used to reset the above, if necessary.
- `write` This event comes just after the switching commands are written to the `aux` file.
- `beforeextras` Just before executing `\extras{\langle language\rangle}`. This event and the next one should not contain language-dependent code (for that, add it to `\extras{\langle language\rangle}`).
- `afterextras` Just after executing `\extras{\langle language\rangle}`. For example, the following deactivates shorthands in all languages:

```
\AddBabelHook{noshort}{afterextras}{\languageshorthands{none}}
```

- `stringprocess` Instead of a parameter, you can manipulate the macro `\BabelString` containing the string to be defined with `\SetString`. For example, to use an expanded version of the string in the definition, write:

```
\AddBabelHook{myhook}{stringprocess}{% \protected@edef\BabelString{\BabelString}}
```

- `initiateactive` (char as active, char as other, original char) New 3.9i Executed just after a shorthand has been 'initiated'. The three parameters are the same character with different catcodes: active, other (\string'ed) and the original one.
- `afterreset` New 3.9i Executed when selecting a language just after `\originalTeX` is run and reset to its base value, before executing `\captions{\langle language\rangle}` and `\date{\langle language\rangle}`.

Four events are used in `hyphen.cfg`, which are handled in a quite different way for efficiency reasons – unlike the precedent ones, they only have a single hook and replace a default definition.

- `everylanguage` (language) Executed before every language patterns are loaded.
- `loadkernel` (file) By default just defines a few basic commands. It can be used to define different versions of them or to load a file.
- `loadpatterns` (patterns file) Loads the patterns file. Used by `luababel.def`. 

45
**loadexceptions** (exceptions file) Loads the exceptions file. Used by luababel.def.

**EXAMPLE** The generic unlocalized \tex hooks are predefined, so that you can write:

\AddToHook{babel/*/afterextras}{\frenchspacing}

which is executed always after the extras for the language being selected (and just before the non-localized hooks defined with \AddBabelHook).

In addition, locale-specific hooks in the form babel/⟨language-name⟩/⟨event-name⟩ are recognized (executed just before the localized babel hooks), but they are not predefined. You have to do it yourself. For example, to set \frenchspacing only in bengali:

\ActivateGenericHook{babel/bengali/afterextras}
\AddToHook{babel/bengali/afterextras}{\frenchspacing}

**\BabelContentsFiles** New 3.9a This macro contains a list of “toc” types requiring a command to switch the language. Its default value is toc, lof, lot, but you may redefine it with \renewcommand (it’s up to you to make sure no toc type is duplicated).

### 1.27 Languages supported by babel with ldf files

In the following table most of the languages supported by babel with and .ldf file are listed, together with the names of the option which you can load babel with for each language. Note this list is open and the current options may be different. It does not include ini files.

- Afrikaans afrikaans
- Azerbaijani azerbaijani
- Basque basque
- Breton breton
- Bulgarian bulgarian
- Catalan catalan
- Croatian croatian
- Czech czech
- Danish danish
- Dutch dutch
- English english, USenglish, american, UKenglish, british, canadian, australian, newzealand
- Esperanto esperanto
- Estonian estonian
- Finnish finnish
- French french, francais, canadien, acadian
- Galician galician
- German austrian, german, germanb, ngerman, naustrian
- Greek greek, polutonikogreek
- Hebrew hebrew
- Icelandic icelandic
- Indonesian indonesian (bahasa, indon, bahasai)
- Interlingua interlingua
- Irish Gaelic irish
- Italian italian
- Latin latin
- Lower Sorbian lowersorbian
- Malay malay, melayu (bahasam)
- North Sami samin
- Norwegian norsk, nynorsk
- Polish polish

46
Portuguese  portuguese, brazilian (portugues, brazil)
Romanian  romanian
Russian  russian
Scottish Gaelic  scottish
Spanish  spanish
Slovakian  slovak
Slovenian  slovene
Swedish  swedish
Turkish  turkish
Ukrainian  ukrainian
Upper Sorbian  uppersorbian
Welsh  welsh

There are more languages not listed above, including hindi, thai, thaićjk, latvian, turkmen, magyar, mongolian, romanish, lithuanian, spanglish, vietnamese, japanese, pinyin, arabic, farsi, ibygreek, bgreek, serbianc, frenchie, ethiop and friulian.
Most of them work out of the box, but some may require extra fonts, encoding files, a preprocessor or even a complete framework (like CJK or luatexja). For example, if you have got the velthuis/devnag package, you can create a file with extension .dn:

\documentclass{article}
\usepackage[hindi]{babel}
\begin{document}
{\dn devaanaa.m priya.h}
\end{document}

Then you preprocess it with devnag \textit{file}, which creates \textit{file}.tex; you can then typeset the latter with \LaTeX.

1.28 Unicode character properties in \textit{luatex}

\textbf{New 3.32} Part of the babel job is to apply Unicode rules to some script-specific features based on some properties. Currently, they are 3, namely, direction (ie, bidi class), mirroring glyphs, and line breaking for CJK scripts. These properties are stored in lua tables, which you can modify with the following macro (for example, to set them for glyphs in the PUA).

\begin{verbatim}
{⟨char-code⟩}{⟨to-char-code⟩}{⟨property⟩}{⟨value⟩}
\end{verbatim}

\textbf{New 3.32} Here, \textit{⟨char-code⟩} is a number (with \TeX syntax). With the optional argument, you can set a range of values. There are three properties (with a short name, taken from Unicode): direction (bc), mirror (bmg), linebreak (lb). The settings are global, and this command is allowed only in vertical mode (the preamble or between paragraphs).

For example:

\begin{verbatim}
\babelcharproperty{'?}{mirror}{'}
\babelcharproperty{'-'}{direction}{l} % or al, r, en, an, on, et, cs
\babelcharproperty{''}){linebreak}{cl} % or id, op, cl, ns, ex, in, hy
\end{verbatim}

\textbf{New 3.39} Another property is locale, which adds characters to the list used by onchar in \texttt{\textbackslash babelprovide}, or, if the last argument is empty, removes them. The last argument is the locale name:

\begin{verbatim}
\babelcharproperty{',}{locale}{english}
\end{verbatim}

1.29 Tweaking some features

\footnote{The two last name comes from the times when they had to be shortened to 8 characters}
Sometimes you might need to disable some babel features. Currently this macro understands the following keys (and only for \latex), with values on or off: \texttt{bidi.text}, \texttt{bidi.mirroring}, \texttt{bidi.mapdigits}, \texttt{layout.lists}, \texttt{layout.tabular}, \texttt{linebreak.sea}, \texttt{linebreak.cjk}, \texttt{justify.arabic}. For example, you can set \texttt{\babeladjust\{bidi.text=off\}} if you are using an alternative algorithm or with large sections not requiring it. Use with care, because these options do not deactivate other related options (like paragraph direction with \texttt{bidi.text}).

1.30 Tips, workarounds, known issues and notes

- If you use the document class \texttt{book} and you use \texttt{\ref} inside the argument of \texttt{\chapter} (or just use \texttt{\ref} inside \texttt{\MakeUppercase\{chapter\}}, \LaTeX{} will keep complaining about an undefined label. To prevent such problems, you can revert to using uppercase labels, you can use \texttt{\lowercase\{\ref\{foo\}\}} inside the argument of \texttt{\chapter}, or, if you will not use shorthands in labels, set the \texttt{safe} option to \texttt{none} or \texttt{bib}.

- Both \texttt{ltxdoc} and babel use \texttt{\AtBeginDocument} to change some catcodes, and babel reloads \texttt{hhline} to make sure: has the right one, so if you want to change the catcode of \texttt{|} it has to be done using the same method at the proper place, with

\begin{verbatim}
\AtBeginDocument{\DeleteShortVerb{|}}
\end{verbatim}

\textit{before} loading babel. This way, when the document begins the sequence is (1) make | active (ltxdoc); (2) make it unactive (your settings); (3) make babel shorthands active (babel); (4) reload \texttt{hhline} (babel, now with the correct catcodes for | and :).

- Documents with several input encodings are not frequent, but sometimes are useful. You can set different encodings for different languages as the following example shows:

\begin{verbatim}
\addto\extrasfrench{\inputencoding{latin1}}
\addto\extrasrussian{\inputencoding{koi8-r}}
\end{verbatim}

- For the hyphenation to work correctly, \texttt{lccodes} cannot change, because \LaTeX{} only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.\footnote{This explains why \LaTeX{} assumes the lowercase mapping of \texttt{T1} and does not provide a tool for multiple mappings. Unfortunately, \texttt{\savinghyphcodes} is not a solution either, because lccodes for hyphenation are frozen in the format and cannot be changed.} So, if you write a chunk of French text with \texttt{\foreignlanguage}, the apostrophes might not be taken into account. This is a limitation of \LaTeX{}, not of babel. Alternatively, you may use \texttt{\useshorthands} to activate \texttt{'} and \texttt{\defineshorthand}, or redefine \texttt{\textquoteright} (the latter is called by the non-ASCII right quote).

- \texttt{\bibitem} is out of sync with \texttt{\selectlanguage} in the .aux file. The reason is \texttt{\bibitem} uses \texttt{\immediate} (and others, in fact), while \texttt{\selectlanguage} doesn't. There is a similar issue with floats, too. There is no known workaround.

- Babel does not take into account \texttt{\normalsf} codes and (non-)French spacing is not always properly (un)set by languages. However, problems are unlikely to happen and therefore this part remains untouched in version 3.9 (but it is in the ‘to do’ list).

- Using a character mathematically active (i.e, with math code “8000) as a shorthand can make \LaTeX{} enter in an infinite loop in some rare cases. (Another issue in the ‘to do’ list, although there is a partial solution.)

The following packages can be useful, too (the list is still far from complete):

\begin{itemize}
  \item \texttt{csquotes} Logical markup for quotes.
\end{itemize}
iflang Tests correctly the current language.
hyphsubst Selects a different set of patterns for a language.
translator An open platform for packages that need to be localized.
siunitx Typesetting of numbers and physical quantities.
biblatex Programmable bibliographies and citations.
bicaption Bilingual captions.
babelbib Multilingual bibliographies.
microtype Adjusts the typesetting according to some languages (kerning and spacing).
Ligatures can be disabled.
substitutefont Combines fonts in several encodings.
mkpattern Generates hyphenation patterns.
tracklang Tracks which languages have been requested.
ucharclasses (xetex) Switches fonts when you switch from one Unicode block to another.
zhspacing Spacing for CJK documents in xetex.

1.31 Current and future work

The current work is focused on the so-called complex scripts in luatex. In 8-bit engines, babel provided a basic support for bidi text as part of the style for Hebrew, but it is somewhat unsatisfactory and internally replaces some hardwired commands by other hardwired commands (generic changes would be much better). Useful additions would be, for example, time, currency, addresses and personal names.\footnote{See for example POSIX, ISO 14652 and the Unicode Common Locale Data Repository (CLDR). Those systems, however, have limited application to \TeX{} because their aim is just to display information and not fine typesetting.}
But that is the easy part, because they don’t require modifying the \LaTeX{} internals. Calendars (Arabic, Persian, Indic, etc.) are under study. Also interesting are differences in the sentence structure or related to it. For example, in Basque the number precedes the name (including chapters), in Hungarian “from (1)” is “(1)-ből”, but “from (3)” is “(3)-ből”, in Spanish an item labelled “3.” may be referred to as either “item 3.” or “3.°” item”, and so on.
An option to manage bidirectional document layout in luatex (lists, footnotes, etc.) is almost finished, but xetex required more work. Unfortunately, proper support for xetex requires patching somehow lots of macros and packages (and some issues related to \specials{} remain, like color and hyperlinks), so babel resorts to the bidi package (by Vafa Khalighi). See the babel repository for a small example (xe-bidi).

1.32 Tentative and experimental code

See the code section for \texttt{\foreignlanguage*} (a new starred version of \texttt{\foreignlanguage}). For old and deprecated functions, see the babel site.

Options for locales loaded on the fly

\texttt{\babeladjust{ autoload.options = ... }} sets the options when a language is loaded on the fly (by default, no options). A typical value would be \texttt{import}, which defines captions, date, numerals, etc., but ignores the code in the \texttt{tex} file (for example, extended numerals in Greek).

Labels

\texttt{\new 3.48} There is some work in progress for babel to deal with labels, both with the relation to captions (chapters, part), and how counters are used to define them. It is still somewhat tentative because it is far from trivial – see the babel site for further details.

2 Loading languages with language.dat

\TeX{} and most engines based on it (pdf\TeX{}, xetex, \epsilon\TeX{}, the main exception being luatex) require hyphenation patterns to be preloaded when a format is created (eg, \LaTeX{}, Xe\LaTeX{}, pdf\LaTeX{}). babel provides a tool which has become standard in many distributions and based on a “configuration file” named \texttt{language.dat}. The exact way this file is used
depends on the distribution, so please, read the documentation for the latter (note also some distributions generate the file with some tool).

**New 3.9q** With luatex, however, patterns are loaded on the fly when requested by the language (except the “0th” language, typically english, which is preloaded always). Until 3.9n, this task was delegated to the package luatex-hyphen, by Khaled Hosny, Élie Roux, and Manuel Pégorié-Gonnard, and required an extra file named language.dat.lua, but now a new mechanism has been devised based solely on language.dat. You must **rebuild the formats** if upgrading from a previous version. You may want to have a local `language.dat` for a particular project (for example, a book on Chemistry).

### 2.1 Format

In that file the person who maintains a TeX environment has to record for which languages he has hyphenation patterns and in which files these are stored. When hyphenation exceptions are stored in a separate file this can be indicated by naming that file after the file with the hyphenation patterns.

The file can contain empty lines and comments, as well as lines which start with an equals (=) sign. Such a line will instruct luatex that the hyphenation patterns just processed have to be known under an alternative name. Here is an example:

```plaintext
% File : language.dat
% Purpose : tell iniTeX what files with patterns to load.
english  english.hyphenations
=british
dutch    hyphen.dutch exceptions.dutch % Nederlands
german   hyphen.ger
```

You may also set the font encoding the patterns are intended for by following the language name by a colon and the encoding code. For example:

```plaintext
german:T1 hyphenT1.ger
german  hyphen.ger
```

With the previous settings, if the encoding when the language is selected is T1 then the patterns in `hyphenT1.ger` are used, but otherwise use those in `hyphen.ger` (note the encoding can be set in `\extras⟨lang⟩`).

A typical error when using babel is the following:

```plaintext
No hyphenation patterns were preloaded for
the language `<lang>' into the format.
Please, configure your TeX system to add them and
rebuild the format. Now I will use the patterns
preloaded for english instead}
```

It simply means you must reconfigure `language.dat`, either by hand or with the tools provided by your distribution.

### 3 The interface between the core of babel and the language definition files

The **language definition files** (ldf) must conform to a number of conventions, because these files have to fill in the gaps left by the common code in babel.def, i.e., the definitions of

---

23 This feature was added to 3.9o, but it was buggy. Both 3.9o and 3.9p are deprecated.
24 The loader for luatex is slightly different as it's not based on babel but on etex.src. Until 3.9p it just didn't work, but thanks to the new code it works by reloading the data in the babel way, i.e., with language.dat.
25 This is because different operating systems sometimes use very different file-naming conventions.
26 This is not a new feature, but in former versions it didn't work correctly.
the macros that produce texts. Also the language-switching possibility which has been
built into the babel system has its implications.

The following assumptions are made:

- Some of the language-specific definitions might be used by plain \TeX users, so the files
  have to be coded so that they can be read by both \LaTeX and plain \TeX. The current
  format can be checked by looking at the value of the macro \texttt{\fmtname}.

- The common part of the babel system redefines a number of macros and environments
  (defined previously in the document style) to put in the names of macros that replace
  the previously hard-wired texts. These macros have to be defined in the language
  definition files.

- The language definition files must define five macros, used to activate and deactivate
  the language-specific definitions. These macros are \texttt{\langle lang \rangle hyphenmins},
  \texttt{\langle lang \rangle captions}, \texttt{\langle lang \rangle date}, \texttt{\langle lang \rangle extras} and
  \texttt{\langle lang \rangle noextras} (the last two may be left empty); where \texttt{\langle lang \rangle}
  is either the name of the language definition file or the name of the \LaTeX option that is to be used. These macros and their functions are
discussed below. You must define all or none for a language (or a dialect); defining, say,
\texttt{\langle lang \rangle date} but not \texttt{\langle lang \rangle captions} does not raise an error but can lead to
unexpected results.

- When a language definition file is loaded, it can define \texttt{\l@\langle lang \rangle} to be a dialect of
  \texttt{\language0} when \texttt{\l@\langle lang \rangle} is undefined.

- Language names must be all lowercase. If an unknown language is selected, babel will
  attempt setting it after lowercasing its name.

- The semantics of modifiers is not defined (on purpose). In most cases, they will just be
  simple separated options (e.g., \texttt{spanish}), but a language might require, say, a set of
  options organized as a tree with suboptions (in such a case, the recommended
  separator is \texttt{/}).

Some recommendations:

- The preferred shorthand is \texttt{"}, which is not used in \LaTeX (quotes are entered as \texttt{`` \textquoteright \textquoteright} and
  \texttt{'' \textquoteright \textquoteright}). Other good choices are characters which are not used in a certain context (e.g., \texttt{=} in
  an ancient language). Note however \texttt{=, <, >, :} and the like can be dangerous, because
  they may be used as part of the syntax of some elements (numeric expressions,
  key/value pairs, etc.).

- Captions should not contain shorthands or encoding-dependent commands (the latter is
  not always possible, but should be clearly documented). They should be defined using
  the LICR. You may also use the new tools for encoded strings, described below.

- Avoid adding things to \texttt{\noextras\langle lang \rangle} except for \texttt{umlaut} and \texttt{high and friends},
  \texttt{\bbl@deactivate}, \texttt{\bbl@(non)fr} \texttt{enchspacing}, and language-specific macros. Use
  always, if possible, \texttt{\bbl@save} and \texttt{\bbl@savevariable} (except if you still want to have
  access to the previous value). Do not reset a macro or a setting to a hardcoded value.
  Never. Instead save its value in \texttt{\extras\langle lang \rangle}.

- Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the
  font encoding (low-level) or the language (high-level, which in turn may switch the font
  encoding). Usage of things like \texttt{\latintext} is deprecated.\footnote{But not removed, for backward compatibility.}

- Please, for “private” internal macros do not use the \texttt{\bbl@} prefix. It is used by babel and
  it can lead to incompatibilities.

There are no special requirements for documenting your language files. Now they are not
included in the base babel manual, so provide a standalone document suited for your
needs, as well as other files you think can be useful. A PDF and a “readme” are strongly
recommended.
3.1 Guidelines for contributed languages

Currently, the easiest way to contribute a new language is by taking one of the 500 or so templates available on GitHub as a basis. Just make a pull request to download it and then, after filling the fields, send it to me. Feel free to ask for help or to make feature requests.

As to ldf files, now language files are “outsourced” and are located in a separate directory (/macros/latex/contrib/babel-contrib), so that they are contributed directly to CTAN (please, do not send to me language styles just to upload them to CTAN).

Of course, placing your style files in this directory is not mandatory, but if you want to do it, here are a few guidelines.

- Do not hesitate stating on the file heads you are the author and the maintainer, if you actually are. There is no need to state the babel maintainer(s) as authors if they have not contributed significantly to your language files.
- Fonts are not strictly part of a language, so they are best placed in the corresponding TeX tree. This includes not only tfm, vf, ps1, otf, mf files and the like, but also fd ones.
- Font and input encodings are usually best placed in the corresponding tree, too, but sometimes they belong more naturally to the babel style. Note you may also need to define a LICR.
- Babel ldf files may just interface a framework, as it happens often with Oriental languages/scripts. This framework is best placed in its own directory.

The following page provides a starting point for ldf files:

If you need further assistance and technical advice in the development of language styles, I am willing to help you. And of course, you can make any suggestion you like.

3.2 Basic macros

In the core of the babel system, several macros are defined for use in language definition files. Their purpose is to make a new language known. The first two are related to hyphenation patterns.

\addlanguage

The macro `\addlanguage` is a non-outerversion of the macro `\newlanguage`, defined in plain.tex version 3.x. Here “language” is used in the TeX sense of set of hyphenation patterns.

\adddialect

The macro `\adddialect` can be used when two languages can (or must) use the same hyphenation patterns. This can also be useful for languages for which no patterns are preloaded in the format. In such cases the default behavior of the babel system is to define this language as a ‘dialect’ of the language for which the patterns were loaded as \language0. Here “language” is used in the TeX sense of set of hyphenation patterns.

\langle lang⟩hyphenmins

The macro `\langle lang⟩hyphenmins` is used to store the values of the \lefthyphenmin and \righthyphenmin. Redefine this macro to set your own values, with two numbers corresponding to these two parameters. For example:

\renewcommand\spanishhyphenmins{34}

(Assigning \lefthyphenmin and \righthyphenmin directly in \extras⟨lang⟩ has no effect.)

\providehyphenmins

The macro `\providehyphenmins` should be used in the language definition files to set \lefthyphenmin and \righthyphenmin. This macro will check whether these parameters were provided by the hyphenation file before it takes any action. If these values have been already set, this command is ignored (currently, default pattern files do not set them).

\captions⟨lang⟩

The macro `\captions⟨lang⟩` defines the macros that hold the texts to replace the original hard-wired texts.
The macro \date{lang} defines \today.

The macro \extras{lang} contains all the extra definitions needed for a specific language. This macro, like the following, is a hook – you can add things to it, but it must not be used directly.

Because we want to let the user switch between languages, but we do not know what state \TeX{} might be in after the execution of \extras{lang}, a macro that brings \TeX{} into a predefined state is needed. It will be no surprise that the name of this macro is \noextras{lang}.

This is a command to be used in the language definition files for declaring a language attribute. It takes three arguments: the name of the language, the attribute to be defined, and the code to be executed when the attribute is to be used.

To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use \mainlanguage instead of \selectlanguage. This will just store the name of the language, and the proper language will be activated at the start of the document.

The macro \ProvidesLanguage should be used to identify the language definition files. Its syntax is similar to the syntax of the \ProvidesPackage command.

The macro \LdfInit performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the @-sign, preventing the .ldf file from being processed twice, etc.

The macro \ldf@quit does work needed if a .ldf file was processed earlier. This includes resetting the category code of the @-sign, preparing the language to be activated at \begin{document} time, and ending the input stream.

The macro \ldf@finish does work needed at the end of each .ldf file. This includes resetting the category code of the @-sign, loading a local configuration file, and preparing the language to be activated at \begin{document} time.

After processing a language definition file, \TeX{} can be instructed to load a local configuration file. This file can, for instance, be used to add strings to \captions{lang} to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by \ldf@finish.

(Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This .fd file will instruct \TeX{} to use a font from the second family when a font from the first family in the given encoding seems to be needed.

### 3.3 Skeleton

Here is the basic structure of an ldf file, with a language, a dialect and an attribute. Strings are best defined using the method explained in sec. 3.8 (babel 3.9 and later).

```latex
\ProvidesLanguage{<language>}
[2016/04/23 v0.0 <Language> support from the babel system]
\LdfInit{<language>}{captions<language>}
\ifdef\undefined\l@<language>\noopatterns{<Language>}
\add
dialect\l@<language>0\fi
\add
dialect\l@<dialect>\l@<language>\bbl@declare@ttribute{<language>}{<attrib>}{\let\captions<language>\captions<attrib><language>}{\providehyphenmins{<language>}{\tw@\thr@@}}
```
NOTE If for some reason you want to load a package in your style, you should be aware it cannot be done directly in the ldf file, but it can be delayed with \AtEndOfPackage. Macros from external packages can be used inside definitions in the ldf itself (for example, \extras{language}), but if executed directly, the code must be placed inside \AtEndOfPackage. A trivial example illustrating these points is:

\AtEndOfPackage{%
  \RequirePackage{dingbat}% Delay package
  \savebox{\myeye}{\eye}% And direct usage
  \newsavebox{\myeye}
  \newcommand{\myanchor}{\anchor}% But OK inside command
\}

3.4 Support for active characters

In quite a number of language definition files, active characters are introduced. To facilitate this, some support macros are provided.

\initiate@active@char
The internal macro \initiate@active@char is used in language definition files to instruct \LaTeX{} to give a character the category code ‘active’. When a character has been made active it will remain that way until the end of the document. Its definition may vary.

\bbl@activate
\bbl@deactivate
The command \bbl@activate is used to change the way an active character expands. \bbl@activate ‘switches on’ the active behavior of the character. \bbl@deactivate lets the active character expand to its former (mostly) non-active self.

\declare@shorthand
The macro \declare@shorthand is used to define the various shorthands. It takes three arguments: the name for the collection of shorthands this definition belongs to; the character (sequence) that makes up the shorthand, i.e. ~ or "a; and the code to be executed when the shorthand is encountered. (It does not raise an error if the shorthand character has not been “initiated”.)

\bbl@add@special
\bbl@remove@special
The \TeX{}book states: “Plain \TeX{} includes a macro called \dospecials that is essentially a set macro, representing the set of all characters that have a special category code.” [4, p. 380] It is used to set text ‘verbatim’. To make this work if more characters get a special category code, you have to add this character to the macro \dospecials. \TeX{} adds another macro called \@sanitize representing the same character set, but without the curly braces. The macros \bbl@add@special{char} and \bbl@remove@special{char} add and remove the character \texttt{char} to these two sets.
3.5 Support for saving macro definitions

Language definition files may want to redefine macros that already exist. Therefore a mechanism for saving (and restoring) the original definition of those macros is provided. We provide two macros for this.\footnote{This mechanism was introduced by Bernd Raichle.}

\texttt{\textbackslash babel@save} To save the current meaning of any control sequence, the macro \texttt{\textbackslash babel@save} is provided. It takes one argument, \texttt{\langle csname\rangle}, the control sequence for which the meaning has to be saved.

\texttt{\textbackslash babel@savevariable} A second macro is provided to save the current value of a variable. In this context, anything that is allowed after the \texttt{\textbackslash the} primitive is considered to be a variable. The macro takes one argument, the \texttt{\langle variable\rangle}.

The effect of the preceding macros is to append a piece of code to the current definition of \texttt{\textbackslash originalTeX}. When \texttt{\textbackslash originalTeX} is expanded, this code restores the previous definition of the control sequence or the previous value of the variable.

3.6 Support for extending macros

\texttt{\addto} The macro \texttt{\addto\{\langle control sequence\rangle\}\{\langle \TeX\ code\rangle\}} can be used to extend the definition of a macro. The macro need not be defined (ie, it can be undefined or \texttt{\textbackslash relax}). This macro can, for instance, be used in adding instructions to a macro like \texttt{\textbackslash extrasenglish}.

Be careful when using this macro, because depending on the case the assignment can be either global (usually) or local (sometimes). That does not seem very consistent, but this behavior is preserved for backward compatibility. If you are using \texttt{etoolbox}, by Philipp Lehman, consider using the tools provided by this package instead of \texttt{\addto}.

3.7 Macros common to a number of languages

\texttt{\textbackslash allowhyphens} In several languages compound words are used. This means that when \texttt{\TeX} has to hyphenate such a compound word, it only does so at the ‘-’ that is used in such words. To allow hyphenation in the rest of such a compound word, the macro \texttt{\textbackslash allowhyphens} can be used.

\texttt{\allowhyphens} Same as \texttt{\allowhyphens}, but does nothing if the encoding is \texttt{T1}. It is intended mainly for characters provided as real glyphs by this encoding but constructed with \texttt{\accent} in \texttt{OT1}.

Note the previous command (\texttt{\allowhyphens}) has different applications (hyphens and discretionary) than this one (composite chars). Note also prior to version 3.7, \texttt{\allowhyphens} had the behavior of \texttt{\bb@allowhyphens}.

\texttt{\set@low@box} For some languages, quotes need to be lowered to the baseline. For this purpose the macro \texttt{\set@low@box} is available. It takes one argument and puts that argument in an \texttt{\hbox}, at the baseline. The result is available in \texttt{\box0} for further processing.

\texttt{\save@sf@q} Sometimes it is necessary to preserve the \texttt{\spacefactor}. For this purpose the macro \texttt{\save@sf@q} is available. It takes one argument, saves the current spacefactor, executes the argument, and restores the spacefactor.

\texttt{\bb@frenchspacing} The commands \texttt{\bb@frenchspacing} and \texttt{\bb@nonfrenchspacing} can be used to properly switch French spacing on and off.

3.8 Encoding-dependent strings

\texttt{New 3.9a} Babel 3.9 provides a way of defining strings in several encodings, intended mainly for \texttt{luatex} and \texttt{xetex}. This is the only new feature requiring changes in language files if you want to make use of it.

Furthermore, it must be activated explicitly, with the package option \texttt{strings}. If there is no \texttt{strings}, these blocks are ignored, except \texttt{\SetCase} (and except if forced as described below). In other words, the old way of defining/switching strings still works and it's used by default.

It consist is a series of blocks started with \texttt{\StartBabelCommands}. The last block is closed with \texttt{\EndBabelCommands}. Each block is a single group (ie, local declarations apply until
the next \StartBabelCommands or \EndBabelCommands. An ldf may contain several series of this kind.

Thanks to this new feature, string values and string language switching are not mixed any more. No need of \addto. If the language is french, just redefine \frenchchaptername.

\StartBabelCommands{\langle language-list\rangle}{\langle category\rangle}[\langle selector\rangle]

The \langle language-list\rangle specifies which languages the block is intended for. A block is taken into account only if the \CurrentOption is listed here. Alternatively, you can define \BabelLanguages to a comma-separated list of languages to be defined (if undefined, \StartBabelCommands sets it to \CurrentOption). You may write \CurrentOption as the language, but this is discouraged – a explicit name (or names) is much better and clearer. A “selector” is a name to be used as value in package option strings, optionally followed by extra info about the encodings to be used. The name unicode must be used for xetex and luatex (the key strings has also other two special values: generic and encoded).

If a string is set several times (because several blocks are read), the first one takes precedence (i.e., it works much like \providecommand). Encoding info is charset= followed by a charset, which if given sets how the strings should be translated to the internal representation used by the engine, typically utf8, which is the only value supported currently (default is no translations). Note charset is applied by luatex and xetex when reading the file, not when the macro or string is used in the document.

A list of font encodings which the strings are expected to work with can be given after fontenc= (separated with spaces, if two or more) – recommended, but not mandatory, although blocks without this key are not taken into account if you have requested strings=encoded.

Blocks without a selector are read always if the key strings has been used. They provide fallback values, and therefore must be the last blocks; they should be provided always if possible and all strings should be defined somehow inside it; they can be the only blocks (mainly LGC scripts using the LICR). Blocks without a selector can be activated explicitly with strings=generic (no block is taken into account except those). With strings=encoded, strings in those blocks are set as default (internally, ?). With strings=encoded strings are protected, but they are correctly expanded in \MakeUppercase and the like. If there is no key strings, string definitions are ignored, but \SetCases are still honored (in a encoded way).

The \langle category\rangle is either captions, date or extras. You must stick to these three categories, even if no error is raised when using other name.\footnote{It may be empty, too, but in such a case using \SetString is an error (but not \SetCase).}

\StartBabelCommands{\langle language\rangle}{\langle captions\rangle}
\[unicode, fontenc=TU EU1 EU2, charset=utf8\]
\SetString{\langle chaptername\rangle}{utf8-string}

\StartBabelCommands{\langle language\rangle}{\langle captions\rangle}
\[unicode, fontenc=TU EU1 EU2, charset=utf8\]
\SetString{\langle chaptername\rangle}{ascii-maybe-LICR-string}

\EndBabelCommands

A real example is:

\StartBabelCommands{austrian}{date}
\[unicode, fontenc=TU EU1 EU2, charset=utf8\]
\SetString{\langle monthiname\rangle}{Jänner}

\StartBabelCommands{german,austrian}{date}
\[unicode, fontenc=TU EU1 EU2, charset=utf8\]

\footnote{In future releases further categories may be added.}
When used in LDF files, previous values of \(\langle\text{category}\rangle\langle\text{language}\rangle\) are overridden, which means the old way to define strings still works and used by default (to be precise, is first set to undefined and then strings are added). However, when used in the preamble or in a package, new settings are added to the previous ones, if the language exists (in the babel sense, ie, if \texttt{\stringdate\langle\text{language}\rangle} exists).

\verb|\{\langle language-list\rangle\}\{\langle category\rangle\}\{\langle selector\rangle\}| \text{The starred version just forces strings to take a value – if not set as package option, then the default for the engine is used. This is not done by default to prevent backward incompatibilities, but if you are creating a new language this version is better. It’s up to the maintainers of the current languages to decide if using it is appropriate.}\footnote{This replaces in 3.9g a short-lived \texttt{\usestrings} which has been removed because it did not work.}

\verb|\EndBabelCommands| Marks the end of the series of blocks.

\verb|\AfterBabelCommands| \texttt{(code)} The code is delayed and executed at the global scope just after \texttt{\EndBabelCommands}.

\verb|\SetString| \texttt{\langle macro-name\rangle}\{\langle string\rangle\} Adds \texttt{\langle\text{macro-name}\rangle} to the current category, and defines globally \texttt{\langle\text{lang-macro-name}\rangle} to \texttt{\langle\text{code}\rangle} (after applying the transformation corresponding to the current charset or defined with the hook \texttt{\stringprocess}). Use this command to define strings, without including any “logic” if possible, which should be a separated macro. See the example above for the date.

\verb|\SetStringLoop| \texttt{\langle macro-name\rangle}\{\langle string-list\rangle\}
A convenient way to define several ordered names at once. For example, to define \abmoniname, \abmoniiname, etc. (and similarly with abday):

\SetStringLoop{abmon#1name}{en, fb, mr, ab, my, jn, jl, ag, sp, oc, nv, dc}
\SetStringLoop{abday#1name}{lu, ma, mi, ju, vi, sa, do}

#1 is replaced by the roman numeral.

\SetCase
{ \langle map-list \rangle \{ \langle to-upper-code \rangle \}\{ \langle to-lower-code \rangle \} } 

Sets globally code to be executed at \MakeUppercase and \MakeLowercase. The code would typically be things like \let \BB \bb and \uccode or \lccode (although for the reasons explained above, changes in lc/uc codes may not work). A \map-list is a series of macros using the internal format of \@uc1clist (eg, \bb\BB\cc\CC). The mandatory arguments take precedence over the optional one. This command, unlike \SetString, is executed always (even without strings), and it is intended for minor readjustments only. For example, as T1 is the default case mapping in \LaTeX, we can set for Turkish:

\StartBabelCommands{turkish}{} [ot1enc, fontenc=OT1]
\SetCase
{ \uccode "I=1\relax }
{ \lccode "I=10\relax }
\EndBabelCommands

(Nota the mapping for OT1 is not complete.)

\SetHyphenMap
{ \langle to-lower-macros \rangle } 

\textbf{New 3.9g} Case mapping serves in \TeX for two unrelated purposes: case transforms (upper/lower) and hyphenation. \SetCase handles the former, while hyphenation is handled by \SetHyphenMap and controlled with the package option hyphenmap. So, even if internally they are based on the same \TeX primitive (\lccode), babel sets them separately.

There are three helper macros to be used inside \SetHyphenMap:

- \BabelLower{ \langle uccode \rangle }{ \langle lccode \rangle } is similar to \lccode but it's ignored if the char has been set and saves the original lccode to restore it when switching the language (except with hyphenmap=first).

- \BabelLowerMM{ \langle uccode-from \rangle }{ \langle uccode-to \rangle }{ \langle step \rangle }{ \langle lccode-from \rangle } loops though the given uppercase codes, using the step, and assigns them the lccode, which is also increased (MM stands for \textit{many-to-many}).

- \BabelLowerMO{ \langle uccode-from \rangle }{ \langle uccode-to \rangle }{ \langle step \rangle }{ \langle lccode \rangle } loops though the given uppercase codes, using the step, and assigns them the lccode, which is fixed (MO stands for \textit{many-to-one}).
An example is (which is redundant, because these assignments are done by both \texttt{luatex} and \texttt{xetex}):

\SetHyphenMap{\texttt{BabelLowerMM}{"100}{"11F}{"101}}

This macro is not intended to fix wrong mappings done by Unicode (which are the default in both \texttt{xetex} and \texttt{luatex}) – if an assignment is wrong, fix it directly.

### 3.9 Executing code based on the selector

\IfBabelSelectorTF{⟨selectors⟩}{{⟨true⟩}}{{⟨false⟩}}

\textbf{New 3.67} Sometimes a different setup is desired depending on the selector used. Values allowed in ⟨selectors⟩ are select, other, foreign, other* (and also foreign* for the tentative starred version), and it can consist of a comma-separated list. For example:

\IfBabelSelectorTF{other, other*}{A}{B}

is true with these two environment selectors. Its natural place of use is in hooks or in \texttt{\textbackslash extras\langle language\rangle}.

## Part II

### Source code

\texttt{babel} 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 \texttt{babel} only as documented (except, of course, if you want to explore and test them – you can post suggestions about multilingual issues to kadingira@tug.org on http://tug.org/mailman/listinfo/kadingira).

### 4 Identification and loading of required files

\textit{Code documentation is still under revision.}

The following description is no longer valid, because switch and plain have been merged into \texttt{babel.def}.

The \texttt{babel} package after unpacking consists of the following files:

- \texttt{switch.def} defines macros to set and switch languages.
- \texttt{babel.def} defines the rest of macros. It has tow parts: a generic one and a second one only for \LaTeX.
- \texttt{babel.sty} is the \LaTeX\X package, which set options and load language styles.
- \texttt{plain.def} defines some \LaTeX\X macros required by \texttt{babel.def} and provides a few tools for Plain.
- \texttt{hyphen.cfg} is the file to be used when generating the formats to load hyphenation patterns.

The \texttt{babel} installer extends docstrip with a few “pseudo-guards” to set “variables” used at installation time. They are used with \texttt{<@name>} at the appropriated places in the source code and shown below with \texttt{⟨⟨name⟩⟩}. That brings a little bit of literate programming.

### 5 locale directory

A required component of \texttt{babel} is a set of ini files with basic definitions for about 200 languages. They are distributed as a separate zip file, not packed as dtx. With them, \texttt{babel} will fully support Unicode engines.

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. Latin and polytonic Greek,
and there are no geographic areas in Spanish). Hindi, French, Occitan and Breton will show a warning related to dates. Not all include LICR variants. This is a preliminary documentation.

INI files contain the actual data; TEX files are currently just proxies to the corresponding INI files. Most keys are self-explanatory.

charset the encoding used in the INI file.

version of the INI file

level “version” of the INI specification. which keys are available (they may grow in a compatible way) and how they should be read.

codings a descriptive list of font encodings.

captions section of captions in the file charset

captions.licr same, but in pure ASCII using the LICR

date.long fields are as in the CLDR, but the syntax is different. Anything inside brackets is a date field (EG, MMM for the month name) and anything outside is text. In addition, [ ] is a non-breakable space and [. ] is an abbreviation dot.

Keys may be further qualified in a particular language with a suffix starting with an uppercase letter. It can be just a letter (EG, babel.name.A, babel.name.B) or a name (EG, date.long.Nominative, date.long.Formal, but no language is currently using the latter). Multi-letter qualifiers are forward compatible in the sense they won’t conflict with new “global” keys (which start always with a lowercase case). There is an exception, however: the section counters has been devised to have arbitrary keys, so you can add lowercased keys if you want.

6 Tools

Do not use the following macros in \TEX 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 \TEX 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.

\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.
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\ldots fi statement\textsuperscript{31}. These macros will break if another if\ldots fi statement appears in one of the arguments and it is not enclosed in braces.

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.

This 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.

To check if a macro is defined, we create a new macro, which does the same as \@ifundefined. However, in an \-tex engine, it is based on \ifcsname, which is more efficient, and does not waste memory.
\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{#4}  
\def\bbl@ifset#1#2#3{  
  \bbl@ifunset{#1}{#3}{\bbl@exp{\\bbl@ifblank{#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=#2=#3@nil@nil{#1}}  
    \expandafter\bbl@kvnext  
  \fi}

\def\bbl@forkv@eq#1=#2=#3@nil@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@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=\the\toks@##2}  
    \bbl@afterfi  
  \bbl@replace@aux##2#2%  
  \fi}%  
  \expandafter\bbl@replace@aux#1#2\bbl@nil#2%  
  \edef#1{\the\toks@}}  
\def\bbl@parsedef##1\detokenize{macro:}{% in #1 -> repl #2 by #3  
  \toks@{}  
  \ifx\bbl@nil#2%  
    \toks@{\expandafter{\the\toks@##1}  
  \else  
    \toks@{\expandafter{\the\toks@##1=\the\toks@##2}  
  \bbl@afterfi  
  \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 \relax by ho, then \relax becomes \rho). No checking is done at all, because it is not a general purpose macro, and it is used by babel only when it works (an example where it does not work is in \bbl@TG@@date, and also fails if there are macros with spaces, because they are retokenized). It may change! (or even merged with \bbl@replace; I'm not sure checking the replacement is really necessary or just paranoia).

\def\bbl@TG@@date{  
  \ifx\detokenize\@undefined% Unused macros if old Plain TeX  
    \bbl@exp{\\bbl@parsedef##1\detokenize{macro:}}{#2}{#3}\relax%  
  \def\bbl@tempdef{#1}  
  \def\bbl@tempb{#2}  
  \def\bbl@tempc{#3}  
  \edef\bbl@tempd{\bbl@tempdef\bbl@tempb\bbl@tempc}  
  \edef\bbl@tempd{\the\toks@\detokenize{macro:}}  
  \edef#1{\the\toks@}  
  \edef#2{\the\toks@}  
  \edef#3{\the\toks@}  
  \edef#4{\the\toks@}
\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{{\scantokens{\bbl@tempa\\@namedef{\bbl@stripslash#1}\bbl@tempb{\bbl@tempe}}}}% Expanded an executed below as 'uplevel'\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 \TeX, 1 is \LaTeX, and 2 is \XeTeX. You may use the latter 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\aftergroup\firstoftwo\else\aftergroup\secondoftwo\fi\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\oe\else\OE\fi\}
\expandafter\in@\expandafter\{\expandafter\OE\expandafter\}\expandafter{% 
\ifin@ \
\bbl@afterelse\expandafter\MakeUppercase 
\else \
\bbl@afterfi\expandafter\MakeLowercase 
\fi} 
\else \
\expandafter\@firstofone 
\fi}

An alternative to IfFormatAtLeastTF for old versions. Temporary.

\ifx\IfFormatAtLeastTF\@undefined 
\def\bbl@ifformatlater{\@ifl@t@r\fmtversion} 
\else 
\let\bbl@ifformatlater\IfFormatAtLeastTF 
\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}

\langleBasic macros\rangle

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

\langleMake sure ProvidesFile is defined\rangle ≡ 
\ifx\ProvidesFile\@undefined 
\def\ProvidesFile#1[#2 #3 #4]{% \wlog{File: #1 #4 #3 <#2>} \let\ProvidesFile\@undefined} \fi 
\langleMake sure ProvidesFile is defined\rangle

6.1 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.

\langleDefine core switching macros\rangle ≡ 
\ifx\language\@undefined 
\csname newcount\endcsname\language \else 
\@temptokena{#2} \edef\bbl@tempc{\the\@temptokena\the\toks@} \toks@\expandafter{\bbl@tempc#3} \expandafter\edef\csname extras\languagename\endcsname{\the\toks@} \fi 
\langleDefine core switching macros\rangle

\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.
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.

## 6.2 The Package File (\TeX, babel.sty)

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

```latex
\@ifpackagewith{babel}{silent}
{\let\bb@info\@gobble
 \let\bb@infowarn\@gobble
\begin{group}
 \def\bb@error#1#2{\PackageError{babel}{#1}{#2}}
 \def\bb@warning#1{\GenericWarning{\PackageWarning}{\PackageInfo}{#1}}
 \end{group}
}
```

65
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.

\begin{code}
\catcode`\^^I=12
\@ifpackagewith{babel}{showlanguages}{%
\begin{group}
\def\bbl@elt#1#2#3#4{%\wlog{#2\^^I#1\^^I#3\^^I#4}}%
\wlog{<*languages>}%
\bbl@languages%
\wlog{</languages>}%
\end{group}%}
\def\bbl@elt#1#2#3#4{%\ifnum#2=\z@\gdef\bbl@nulllanguage{#1}%;\def\bbl@elt##1##2##3##4{}%;\fi}%
\bbl@languages%
\@ifpackagewith{babel}{base}{%
\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}{}
\end{group}
\end{code}

6.3 base

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

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

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

6.4 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.

\begin{verbatim}
\bbl@trace{key=value and another general options}
\bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname\endcsname
\def\bbl@tempb#1.#2{% Remove trailing dot
#1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
\endinput}{}%
\end{verbatim}
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.

Handling of package options is done in three passes. (I 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. Then we "flag" valid keys with a nil value. Then we "flag" valid keys with a nil value.

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

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

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

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

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

6.5 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\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).

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

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

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 set.

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

6.6 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 \textit{Emulate \LaTeX}.

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.
7 Multiple languages

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

\begin{verbatim}
\def\bbl@version{{\langle\langle\texttt{version}\rangle\rangle}}
\def\bbl@date{{\langle\langle\texttt{date}\rangle\rangle}}
\end{verbatim}

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

\begin{verbatim}
\def\adddialect#1#2{\
\global\chardef#1\relax
\bbl@usehooks{adddialect}{{#1}{#2}}\
\begingroup
\count@#1\relax
\def\bbl@elt##1##2##3##4{\
\ifnum\count@=##2\relax
\edef\bbl@tempa{\expandafter\@gobbletwo\string#1}\
\bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa' set to \string\language\the\count@. Reported}\n
\def\bbl@elt####1####2####3####4{}\
\fi}\
\bbl@cs{languages}\
\endgroup}
\end{verbatim}

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

\begin{verbatim}
\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{\texttt{\textbackslash bbl@usehooks}{\texttt{languages}}{#1}}}\
\endgroup}
\end{verbatim}

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

\begin{verbatim}
\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{\texttt{\textbackslash bbl@usehooks}{\texttt{languages}}{#1}}}\
\endgroup}
\end{verbatim}

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 \texttt{\textbackslash bbl@bcpcase}, casing is the correct one, so that \texttt{sr-latn-ba} becomes \texttt{fr-Latn-BA}. Note \texttt{#4} may contain some \texttt{\@empty}'s, but they are eventually removed. \texttt{\textbackslash bbl@bcplookup} either returns the found ini or it is \texttt{\textbackslash relax}.

\begin{verbatim}
\def\bbl@bcpcase#1#2#3#4@@#5{%\ifx\empty#3\%\else\uppercase{\def#5(#1#2)}\%\else\lowercase{\def#5(#1)}\%\lowercase{\def#5(#5#2#3#4)}\}%
\end{verbatim}
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}}

\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\csname selectlanguage \endcsname}

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

The following definition is preserved for backwards compatibility (eg, arabic, koma). It is related to a trick for 2.0.9, 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:
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\}% \} % \edef\localeid{bbl@cl{id@}} %

The unprotected part of \selectlanguage.

\expandafter\def\csname selectlanguage \endcsname#1{% \ifnum\bbl@hymapsel=\tw@\let\bbl@hymapsel=\tw@\fi \bbl@push@language \aftergroup\bbl@pop@language \bbl@set@language{#1}}

\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 or \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. \texttt{\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 \texttt{\write} altogether when not needed).

\begin{verbatim}
\def\BabelContentsFiles{toc,lof,lot}
\def\bbl@set@language#1{% from selectlanguage, pop@%
  \edef\languagename{%
    \ifnum\escapechar=\string#1\@empty
      \else\string#1\@empty\fi}
  \ifcat\relax
    \expandafter\ifx\csname date\languagename\endcsname\relax
      \edef\languagename{#1}
      \let\localename\languagename
    \else
      \bbl@info{Using \texttt{\string\language} instead of \texttt{\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
        \bbl@savelastskip
        \protected@write\@auxout{}{\string\babel@aux{\bbl@auxname}{}}%
        \bbl@restorelastskip
      \fi
    \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}%
  \fi
  \ifnum\bbl@hymapsel=\@cclv \chardef\bbl@hymapsel4 \relax\fi
  \edef\languagename{#1}%
  \bbl@fixname\languagename
  \bbl@iflanguage\languagename{%
    % set hymap
    \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%
  \fi
}\end{verbatim}
First, check if the user asks for a known language. If so, update the value of \language and call \originalTeX to bring \TeX in a certain pre-defined state.

The name of the language is stored in the control sequence \languagename. Then we have to redefine \originalTeX to compensate for the things that have been activated. To save memory space for the macro definition of \originalTeX, we construct the control sequence name for the `\noextras` command at definition time by expanding the \csname primitive. Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of \selectlanguage, and calling these macros.

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

\newif\ifbbl@usedategroup
\def\bbl@switch#1{% from select@, foreign@
  \bbl@ensureinfo{#1}%
  \originalTeX
  \expandafter\def\expandafter\originalTeX\expandafter{%
    \csname noextras#1\endcsname
    \let\originalTeX\@empty
  }
  \bbl@beginsave
  \languageshorthands{none}
  \bbl@id@assign
  \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 \langle lang\rangle date
        \csname date#1\endcsname\relax
      \fi
  \fi
}
otherlanguage

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

The \ignorespaces command is necessary to hide the environment when it is entered in horizontal mode.
The `\endotherlanguage` part of the environment tries to hide itself when it is called in horizontal mode.

```latex
\long\def\endotherlanguage{\global\@ignoretrue\ignorespaces}
```

**otherlanguage** 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`.

```latex
\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}\\\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”.

```latex
\expandafter\let\csname endotherlanguage*\endcsname\relax
```

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

\foreignlanguage#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{ %
\expandafter\ifx\csname date\languagename\endcsname\relax %
\bbl@warning % TODO - why a warning, not an error?
{Unknown language '#1'. Either you have\% misspelled its name, it has not been installed,\% or you requested it in a previous run. Fix its name,\% install it or just rerun the file, respectively. In\% some cases, you may need to remove the aux file.\% I'll proceed, but expect wrong results.\%
Reported} %
\fi
% set type
\let\bbl@select@type\@ne %
\expandafter\bbl@switch\expandafter{\languagename}}}

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

\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 \bbl@hyphenation, 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 #1:\@encoding\endcsname\relax
The environment \texttt{hyphenrules} can be used to select \emph{just} the hyphenation rules. This environment does not change the language name and when the hyphenation rules specified were not loaded it has no effect. Note however, \texttt{lccode}'s and font encodings are not set at all, so in most cases you should use \texttt{otherlanguage*}.

\begin{verbatim}
868 \def\hyphenrules#1{%
869 \edef\bbl@tempf{#1}%
870 \bbl@fixname\bbl@tempf
871 \bbl@iflanguage\bbl@tempf{%
872 \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
873 \ifx\languageshorthands\@undefined\else
874 \languageshorthands{none}%
875 \fi
876 \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
877 \set@hyphenmins\tw@\thr@@\relax
878 \else
879 \expandafter\expandafter\expandafter\set@hyphenmins
880 \csname\bbl@tempf hyphenmins\endcsname\relax
881 \fi}
882 \let\endhyphenrules@empty
\end{verbatim}

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

\begin{verbatim}
883 \def\providehyphenmins#1#2{%
884 \expandafter\ifx\csname#1 hyphenmins\endcsname\relax
885 \@namedef{#1 hyphenmins}{#2}%
886 \fi}
\end{verbatim}

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

\begin{verbatim}
887 \def\set@hyphenmins#1#2{%
888 \lefthyphenmin#1\relax
889 \righthyphenmin#2\relax}
\end{verbatim}

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

\begin{verbatim}
890 \iffalse\ProvidesFile@undefined
891 \def\ProvidesLanguage#1[#2 #3 #4]{%
\end{verbatim}
\log{Language: #1 #4 #3 <#2>}%

\else
\def\ProvidesLanguage#1{%
\begingroup
\catcode`\ 10 %
\@makeother/\%
@ifnextchar\[%\]{{@provideslanguage{#1}}}{{@provideslanguage{#1}\[]}}%
\def\@provideslanguage#1\[#2\]{%
\log{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}%
{Find an armchair, sit down and wait}}%
\let\uselocale\setlocale
\let\locale\setlocale
\let\selectlocale\setlocale
\let\textlocale\setlocale
\let\textlanguage\setlocale
\let\languagetext\setlocale

7.2 Errors

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

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, 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.

\@nulllanguage
\@nocaption
\@tentative

\@nolanerr
\@nopatterns
\@noopterr

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.

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, 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.

\def\@nulllanguage{\string\language=0}
\def\@nocaption{\protect\@nocaption@i}
\def\@nocaption@i#1{
\textbf{?#1?}}
\def\@tentative{\protect\@tentative@i}
\def\@tentative@i#1{
\textbf{?#1?}}
Some functions for '\texttt{#1}' are tentative.\%
They might not work as expected and their behavior\%
could change in the future.\%
Reported}}\)
\def\@nolanerr#1{\%
\bbl@error\%
\{You haven't defined the language '\texttt{#1}' yet.\%
Perhaps you misspelled it or your installation\%
is not complete\}%
\{Your command will be ignored, type <return> to proceed\}}\%
\def\@nopatterns#1{\%
\bbl@warning\%
\{No hyphenation patterns were preloaded for\%
the language '\texttt{#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
\ifx\directlua\@undefined\else\%
\ifx\bbl@luapatterns\@undefined\%
\input luababel.def\%
\fi\%
\fi\%
\l@english\z@\%
\fi\%
\addto It takes two arguments, a \texttt{(control sequence)} and \TeX-code to be added to the \texttt{(control sequence)}.\%
If the \texttt{(control sequence)} has not been defined before it is defined now. The control sequence could also expand to \texttt{\relax}, in which case a circular definition results. The net result is a stack overflow. Note there is an inconsistency, because the assignment in the last branch is global.\%
\def\addto#1#2{%\%
\ifx#1\@undefined\%
\def#1{#2}\%
\else\%
\ifx#1\relax\%
\def#1{#2}\%
\else\%
\def\uselanguage#1{}\%
\input language.def\%
\endgroup\%
\fi\%
\fi\%
\ch@r\def\l@english\z@\%
\fi\%
\addto It takes two arguments, a \texttt{(control sequence)} and \TeX-code to be added to the \texttt{(control sequence)}.\%
If the \texttt{(control sequence)} has not been defined before it is defined now. The control sequence could also expand to \texttt{\relax}, in which case a circular definition results. The net result is a stack overflow. Note there is an inconsistency, because the assignment in the last branch is global.\%
\def\addto#1#2{%\%
\ifx#1\@undefined\%
\def#1{#2}\%
\else\%
\ifx#1\relax\%
\def#1{#2}\%
\else\%
\def\uselanguage#1{}\%
\input language.def\%
\endgroup\%
\fi\%
\fi\%
\ch@r\def\l@english\z@\%
\fi
The macro \initiateactivechar below takes all the necessary actions to make its argument a shorthand character. The real work is performed once for each character. But first we define a little tool. TODO. Always used with additional expansions. Move them here? Move the macro to basic?

\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 \orgmacro.
\bbl@redefine#1{\edef\bbl@tempa{\bbl@stripslash#1}\expandafter\let\csname org@\bbl@tempa\endcsname#1\expandafter\def\csname\bbl@tempa\endcsname}
\bbl@redefine@long
This version of \bbl@redefine can be used to redefine \long commands such as \ifthenelse.
\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 \foono.
\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}
\bbl@usehooks#1#2{\ifx\UseHook\@undefined\else\UseHook{babel/*/#1}\fi\def\bbl@elth##1{\mbox{\expandafter\let\csname ##1\endcsname expl#2}}}}\@onlypreamble\bbl@usehooks\bbl@trace{Hooks}
\newcommand\AddBabelHook[3][]{}{\bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}\def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}\expandafter\bbl@tempa\bbl@evargs,#3=,\@empty\bbl@ifunset{bbl@ev@#2@#3@#1}{}{\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}\bbl@csarg\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}
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).

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}

\edef\bbl@tempa{#1=#2\@@}{\AddBabelHook{babel-ensure}{afterextras}{%\iffalse\bbl@select@type\else\fi}%.% Test required for Plain (?)\iffalse\UseHook{babel/\languagename/#1}\fi% \edef\bbl@tempb##1{\ifx\@empty##1\else
  \noexpand##1\expandafter\bbl@tempb\fi}%.% 3.32 - Don't assume the macro exists\edef\bbl@tempa{\bbl@tempb#1\@empty}{% \edef\bbl@tempb##1=##2\@@{\@namedef{bbl@ens@##1}{##2}}%.% 3.32 - Don't assume the macro exists\bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}
\edef\bbl@tempc{\bbl@ensure}%.% 3.32 - Don't assume the macro exists\expandafter\bbl@add\expandafter\bbl@tempc\expandafter{\expandafter{\bbl@ens@include}}%\expandafter{\expandafter{\bbl@ens@exclude}}% \toks@\expandafter{\bbl@tempc}%.% 3.32 - Don't assume the macro exists\bbl@exp{% \iffalse\bbl@select@type\else\fi}% \begin{group}
\let\bbl@ens@include\@empty% \let\bbl@ens@exclude\@empty% \let\bbl@ens@fontenc\relax% \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list% \iffalse\else\expandafter\def\expandafter\bbl@ens@##1{% \bbl@streq\languagename\bbl@ens@##1}}% \fi% \iffalse\else\edef\bbl@ens@##1{%\bbl@streq\languagename\bbl@ens@##1}{}% \fi%\expandafter\fi}%.% 3.32 - Don't assume the macro exists\def\bbl@ens@include{\bbl@tempb}
\def\bbl@ens@exclude{\bbl@tempb}
\def\bbl@ens@fontenc{\bbl@tempb}
\end{group}

The user command just parses the optional argument and creates a new macro \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.
7.4 Setting up language files

\LdfInit macro takes two arguments. The first argument is the name of the language that will be defined in the language definition file; the second argument is either a control sequence or a string from which a control sequence should be constructed. The existence of the control sequence indicates that the file has been processed before. At the start of processing a language definition file we always check the category code of the at-sign. We make sure that it is a 'letter' during the processing of the file. We also save its name as the last called option, even if not loaded.

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

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

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

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

Finally we check \originalTeX.

\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

\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
\ldf@quit{#1}\%
\else
\expandafter\ifx\csname#2\endcsname\relax
\ldf@quit{#1}\%
\fi
\else
\fi
\bbl@ldfinit}
\ldf@quit This macro interrupts the processing of a language definition file.
\def\ldf@quit#1{\%}
\expandafter\main@language\expandafter{#1}\%
\catcode`@=\atcatcode \let\atcatcode\relax
\catcode`==\eqcatcode \let\eqcatcode\relax}
\ldf@finish This macro takes one argument. It is the name of the language that was defined in the language definition file.
We load the local configuration file if one is present, we set the main language (taking into account that the argument might be a control sequence that needs to be expanded) and reset the category code of the @-sign.
\def\bbl@afterldf#1{\%
\bbl@afterlang
\let\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 \TeX.
\@onlypreamble\LdfInit\ldf@finish
\onlypreamble\ldf@quit\ldf@finish
\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 \%
\setlocalename\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 \pagdir, so we set here for the whole document to the main \bodydir.
\def\bbl@beforestart{\%
\def\nolanerr#1{\%
\def\original\fii{\fii}
A bit of optimization. Select in heads/feet the language only if necessary.

\begin{quote}
7.5 Shorthands
\end{quote}

\texttt{\@add@special} The macro \texttt{\@add@special} is used to add a new character (or single character control sequence) to the macro \texttt{\dospecials} (and \texttt{\@sanitize} if \LaTeX{} is used). It is used only at one place, namely when \texttt{\initiate@active@char} is called (which is ignored if the char has been made active before). Because \texttt{\@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 \texttt{\fss@catcodes}, added in 3.10.

\begin{verbatim}
\begin{Verbatim}
\@add@special \% 1:a macro like ",\?, etc.
\@add@singlespecials\dospecials{\do{##1}}% test @sanitize = \relax, for back. compat.
\ifx\@makeother\@undefined\else
\begingroup
\catcode`#1\active
\fss@catcodes
\expandafter\ifnum\catcode`#1=\active
\endgroup
\@add\fss@catcodes\@makeother#1\fi
\else
\endgroup
\fi
\end{Verbatim}

\texttt{\@remove@special} The companion of the former macro is \texttt{\@remove@special}. It removes a character from the set \texttt{\dospecials} and \texttt{\@sanitize}, but it is not used at all in the babel core.

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

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

However, a deactivated shorthand (with \texttt{\bbldeactivate} is defined as \texttt{\activeprefix \normalchar}. The following macro is used to define shorthands in the three levels. It takes 4 arguments: the (string'ed) character, \texttt{\langle level\rangle group, <level>@active} and \texttt{<next-level>@active} (except in system).

\begin{verbatim}
\def\bblactive@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}\
  \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}\
}
\end{verbatim}

\texttt{\initiateactivechar} calls \texttt{\@initiateactivechar} with 3 arguments. All of them are the same character with different catcodes: active, other (\texttt{\string}) and the original one. This trick simplifies the code a lot.

\begin{verbatim}
\def\initiateactivechar#1{\
  \bbl@ifunset{active\string#1}{\bbl@withactive\
    {\expandafter\@initiateactivechar\expandafter}#1\string#1#1}\
}
\end{verbatim}

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 \texttt{\relax} and preserving some degree of protection).

\begin{verbatim}
\def\@initiateactivechar#1#2#3#4{\texttt{\catcode`#2=\the\catcode`#2\relax}}\
\def\ifx\texttt{\undefined}\
\def\@initiateactivechar#1#2#3#4{\if\texttt{\defined\undefined}\
  {\expandafter\@initiateactivechar\expandafter}#1\string#1#1}\
}
\end{verbatim}

If the character is already active we provide the default expansion under this shorthand mechanism. Otherwise we write a message in the transcript file, and define \texttt{\normalchar(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$
\fi$

To prevent problems with the loading of other packages after babel were set 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}\i}%
\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
\if\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\noexpand\csname normal@char#2\endcsname
\noexpand\else
\noexpand\expandafter\noexpand\csname bbl@doactive#2\endcsname
\noexpand\fi}%
\bbl@csarg\edef{doactive#2}{\noexpand\csname user@active#2\endcsname}{%$}

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

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

\bbl@csarg\edef{active@#2}{%}
\noexpand\active@prefix\noexpand\active

88
The next level of the code checks whether a user has defined a shorthand for himself with this character. First we check for a single character shorthand. If that doesn’t exist we check for a shorthand with an argument.

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

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

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

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

This command helps the shorthand supporting macros to select how to proceed. Note that this macro needs to be expandable as do all the shorthand macros in order for them to work in expansion-only environments such as the argument of \hyphenation. This macro expects the name of a group of shorthands in its first argument and a shorthand character in its second argument. It will expand to either \texttt{\bbl@firstcs} or \texttt{\bbl@scndcs}. Hence two more arguments need to follow it.
The command `\active@prefix` which is used in the expansion of active characters has a function similar to `\OT1-cmd` in that it protects the active character whenever `\protect` is not `\typeset@protect`. The `\@gobble` is needed to remove a token such as `\activechar:` (when the double colon was the active character to be dealt with). There are two definitions, depending of `\ifincsname` is available. If there is, the expansion will be more robust.

\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

In some circumstances it is necessary to be able to change the expansion of an active character on the fly. For this purpose the switch `\@safe@actives` is available. The setting of this switch should be checked in the first level expansion of `\active@char\langle char\rangle`.

\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}

Both macros take one argument, like `\initiate@activechar`. The macro is used to change the definition of an active character to expand to `\activechar\langle char\rangle` in the case of `\bbl@activate`, or `\normalchar\langle char\rangle` 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#2}
\def\bbl@scndcs#1#2{%\csname#2\endcsname}
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.

\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.

\textormath

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

\user@group
\language@group
\system@group
\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}}}\@empty}{\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}\def\bbl@usesh@x#1#2{%\bbl@ifshorthand{#2}{\def\user@group{user}\initiate@active@char{#2}\bbl@activate{#2}}{%\bbl@error{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}}}

\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@\protect@\endcsname{\expandafter\noexpand\csname user@active#1\endcsname}}}\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#1\@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 \ First the new shorthand needs to be initialized. Then, we define the new shorthand in terms of the original one, but note with \aliasshorthands("/"{}/) is \active@prefix /\active@char/, so we still need to let the latest to \active@char".\def\aliasshorthand#1{%\bbl@ifshorthand{#2}{\expandafter\ifx\csname active@char\string#2\endcsname\relax\ifx\document\@notprerr\@notshorthand{#2}\else\initiate@active@char{#2}\expandafter\let\csname active@char\string#2\expandafter@endcsname\csname active@char\string#1\endcsname\fi}}
\@notshorthand
\def\notshorthand#1{\bbl@error{\string #1 is not a shorthand.}}
\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.
\shorthandon \def\shorthandon[1]\{\bbl@switch@sh\neq #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": should exist. Switching on and off 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\@sw#2\@nnil\else
\bbl@ifunset{bbl@active@\string #2}\%
\bbl@error{I can’t switch \string #2 on or off--not a shorthand}
\bbl@ifcase#2 off, on, off*
\catcode’#2\relax
\or
\catcode’2\active
\bbl@ifunset{bbl@shdef@\string #2}\%
\%
\bbl@withactive{\expandafter\let\expandafter}{shdef@\string #2}\%
\csname bbl@shdef@\string #2\endcsname
\bbl@csarg\let{shdef@\string #2}\relax}
\ifcase\bbl@activated\or
\bbl@activate\#
\else
\bbl@deactivate\#
\fi
\or
\bbl@ifunset{bbl@shdef@\string #2}\%
\%
\bbl@withactive{\bbl@csarg\let{shdef@\string #2}}\#
\%
\csname bbl@oricat@\string #2\endcsname
\csname bbl@oridef@\string #2\endcsname
\fi\%
\bbl@afterfi\bbl@switch@sh%
Note the value is that at the expansion time; e.g., in the preamble shorthands are usually deactivated.

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

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.

Usually the ~ is active and expands to \penalty\@M \, . When it is written to the .aux file it is written expanded. To prevent that and to be able to use the character ~ as a start character for a shorthand, it is redefined here as a one character shorthand on system level. The system declaration is in most cases redundant (when ~ is still a non-break space), and in some cases is inconvenient (if ~ has been redefined); however, for backward compatibility it is maintained (some existing documents may rely on the babel value).
The position of the double quote character is different for the OT1 and T1 encodings. It will later be selected using the \f@encoding macro. Therefore we define two macros here to store the position of the character in these encodings.

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

\[\text{\expandafter\def\csname OT1dqpos\endcsname{127}}\]
\[\text{\expandafter\def\csname T1dqpos\endcsname{4}}\]

7.6 Language attributes

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

\[\text{\newcommand*{\languageattribute}[2]{}}\]
\[\text{\def\bbl@tempc{#1}}\]
\[\text{\bbl@fixname\bbl@tempc}\]
\[\text{\bbl@iflanguage\bbl@tempc{}}\]
\[\text{\bbl@vforeach{#2}{}}\]

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

\[\text{\ifx\bbl@known@attribs\@undefined} \text{\in@false}\]
\[\text{\else}\]
\[\text{\bbl@xin{,\bbl@tempc-##1,},\bbl@known@attribs,}}\]
\[\text{\fi}\]
\[\text{\ifin@} \text{\bbl@warning{}}\]
\[\text{\You have more than once selected the attribute '##1' for language #1. Reported}}\]
\[\text{\else}\]
\[\text{\bbl@exp{}}\]
\[\text{\bbl@add@list\bbl@known@attribs{\bbl@tempc-##1}}}\]
\[\text{\edef\bbl@tempa{\bbl@tempc-##1}}\]
\[\text{\expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes}{}\]
\[\text{\@attrerr{\bbl@tempc}{##1}}\]
\[\text{\fi}}\]
\[\text{\@onlypreamble\languageattribute}\]

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

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

\bbl@declare@tribute

This command adds the new language/attribute combination to the list of known attributes. Then it defines a control sequence to be executed when the attribute is used in a document. The result of this should be that the macro \extras... for the current language is extended, otherwise the attribute will not work as its code is removed from memory at \begin{document}. 95
\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.

\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.

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

7.7 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 \original\TeX). Note undefined macros are not undefined any more when saved – they are \relax'ed.
The initialization of a new save cycle: reset the counter to zero.

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

The macro \texttt{babel@save(name)} saves the current meaning of the control sequence \texttt{name} to \texttt{originalTeX}. To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to \texttt{originalTeX} and the counter is incremented. The macro \texttt{babel@savevariable(name)} saves the value of the variable. \texttt{name} can be anything allowed after the \texttt{\the} primitive.

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

\texttt{\bbl@frenchspacing} has to be expandable, i.e. you shouldn't let it to \texttt{\relax}.
7.8 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}the\bbl@tempa\bbl@tempa{\bbl@tempb\bbl@tempa\@@}}

7.9 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]{{\if\bbl@hyphenation@\relax\let\bbl@hyphenation@\@empty\fi}{\if\bbl@hyphlist\@empty\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 33.\bbl@allowhyphens\def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}

33\TeX{} begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.
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\babelhypenon{\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}}}\bbl@hy@#1#2\@empty\@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 1df’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{\bbl@usehyphen{\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{\hskip\z@skip}

\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{\nobreak\discretionary{\#{2}\{}{\#{1}}\bbl@allowhyphens}

7.10 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 couple of tools. The first one makes global a local variable. This is not the best solution, but it works.

\@trace{Multiencoding strings}
\def\bbl@toglobal#1{\global\let#1#1}
\def\bbl@recatcode#1{% TODO. Used only once?
\@tempcnta=7F
\def\bbl@tempa{%
\ifnum\@tempcnta>FF\else
\catcode\@tempcnta=\relax
\advance\@tempcnta\@ne
\expandafter\bbl@tempa
\fi\}%
\bbl@tempa}
The second one. We need to patch \@ucclist, but it is done once and only if \SetCase is used or if strings are encoded. The code is far from satisfactory for several reasons, including the fact \@ucclist is not a list any more. Therefore a package option is added to ignore it. Instead of gobbling the macro getting the next two elements (usually \reserved@a), we pass it as argument to \bbl@uclc. The parser is restarted inside \lang@bbl@uclc because we do not know how many expansions are necessary (depends on whether strings are encoded). The last part is tricky – when uppercasing, we have:

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

and starts over (and similarly when lowercasing).
\ifpackagewith{babel}{nocase}%
\def\bbl@patchuclc@empty\relax
\def\bbl@patchuclc{\global\let\bbl@patchuclc\relax}
\g@addto@macro\@ucclist{\reserved@b{\reserved@b\bbl@uclc}}%
\gdef\bbl@uclc##1{%
\let\bbl@encoded\bbl@encoded@uclc
\bbl@ifunset{\languagename \bbl@uclc}{##1}{\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
\csname\languagename \bbl@uclc@endcsname}{\bbl@tolower@empty}{\bbl@toupper@empty}}%
\gdef\bbl@tolower{\csname\languagename \bbl@lc@endcsname}%
\gdef\bbl@toupper{\csname\languagename \bbl@uc@endcsname}}%
\DeclareOption{nocase}{}
\DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
\DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
\def\BabelStringsDefault{generic}
\endinput

Main command  This is the main command. With the first use it is redefined to omit the basic setup in subsequent blocks. We make sure strings contain actual letters in the range 128-255, not active characters.
Parse the encoding info to get the label, input, and font parts. Select the behavior of \SetString. There are two main cases, depending on if there is an optional argument: without it and \strings=encoded, strings are defined always; otherwise, they are set only if they are still undefined (i.e., fallback values). With labelled blocks and \strings=encoded, define the strings, but with another value, define strings only if the current label or font encoding is the value of \strings; otherwise (i.e., no \strings or a block whose label is not in \strings=) do nothing. We presume the current block is not loaded, and therefore set (above) a couple of default values to gobble the arguments. Then, these macros are redefined if necessary according to several parameters.
There are two versions of \bbl@scswitch. The first version is used when ldf\$s are read, and it makes sure \langle group \rangle \langle language \rangle is reset, but only once (\bbl@screset is used to keep track of this). The second version is used in the preamble and packages loaded after babel and does nothing. The macro \bbl@forlang loops \bbl@L but its body is executed only if the value is in \BabelLanguages (inside babel) or \date \langle language \rangle is defined (after babel has been loaded). There are also two version of \bbl@forlang. The first one skips the current iteration if the language is not in \BabelLanguages (used in ldfs), and the second one skips undefined languages (after babel has been loaded).

\def\bbl@forlang#1#2{%  
\bbl@for#1\bbl@L{%  
\bbl@xin@{,#1,},\BabelLanguages,%}
\ifin@#2\relax\fi}
\AtEndOfPackage{%  
\def\bbl@forlang#1#2{%  
\bbl@for#1\bbl@L{%  
\bbl@xin@{,#1,}{,\ifunset{date#1}{}{#2}}}}
\let\bbl@scswitch\relax}
Now we define commands to be used inside \StartBabelCommands.

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

First save the “switcher”. Create it if undefined. Strings are defined only if undefined (i.e., like \providecommand). 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.

\begin{verbatim}
\def\@setstring#1#2{\let\@tempa#1 \edef\LC{\@tempa\stripslash#1} \if\not\@tempa\relax \edef\\LC{\germanchaptername} \fi \def\BabelString{\global\expandafter\add\the\BabelString\LC\stringdef\CSname\LC\stringdef\stringprocess\stringdef\CSname\LC} \expandafter\csname\LC\endcsname{\BabelString}}
\end{verbatim}

Now, some additional stuff to be used when encoded strings are used. Captions then include \bbl@encoded for string to be expanded incase transformations. It is \relax by default, but in \MakeUppercase and \MakeLowercase its value is a modified expandable \@changed@cmd.

\begin{verbatim}
\if\@opt@strings\relax \def\scset#1#2{\def#1{\bbl@encoded#2}} \patchuclc \let\encoded\relax \def\encoded@uclc#1{\TextSymbolUnavailable#1} \else \def\scset#1#2{\def#1{#2}} \fi \end{verbatim}

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.

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

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

\begin{verbatim}
\def\aftercmds\endgroup
\end{verbatim}
Case mapping

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

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

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

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

This section ends with a general tool for resetting the caption names with a unique interface. With the old way, which mixes the switcher and the string, we convert it to the new one, which separates these two steps.
7.11 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}

7.12 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.

7.12.1 Quotation marks

\quotedblbase

In the T1 encoding the opening double quote at the baseline is available as a separate character, accessible via \textquotedblbase. 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 \guillemetrigh t

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{\guillemetrigh t}{OT1}{\% \ifmmode \gg \else \save@sf@q{\nobreak \raise.2ex\hbox{\$\scriptscriptstyle\gg\$}\bbl@allowhyphens}\% \fi}

106
Makes sure that when an encoding other than OT1 or T1 is used these glyphs can still be typeset.

\ProvideTextCommandDefault{\guilsingleleft}{\UseTextSymbol{OT1}{\guilsingleleft}}
\ProvideTextCommandDefault{\guilsinglright}{\UseTextSymbol{OT1}{\guilsinglright}}

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

\DeclareTextCommandDefault{\ij}{\UseTextSymbol{OT1}{\ij}}
\DeclareTextCommandDefault{\IJ}{\UseTextSymbol{OT1}{\IJ}}

7.12.2 Letters

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

\DeclareTextCommand{\ij}{OT1}{\kern-0.02em\bbl@allowhyphens j}
\DeclareTextCommand{\IJ}{OT1}{\kern-0.02em\bbl@allowhyphens J}

Make sure that when an encoding other than OT1 or T1 is used these glyphs can still be typeset.
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@\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
\advance\dimen@ii.15ex % correction for the dash position
\dimen@ii-.15\fontdimen7\font % correction for cmtt font
\dimen@\thr@@\expandafter\rem@pt\the\fontdimen7\font
\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.

\def\SS@\{
\hrule height0.1em width0.3em\}
\DeclareTextCommand{\SS}{OT1}{SS}
\ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\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}}

7.12.3 Shorthands for quotation marks

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

\glq The 'german' single quotes.
\grq
\providecommand{\glq}{% 
\texttt{\textquoteleft}
\texttt{\textquoteleft}}
\providecommand{\grq}{% 
\texttt{\textquotedblleft}
\texttt{\textquotedblleft}}

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

\glqq The 'german' double quotes.
\grqq
\providecommand{\glqq}{% 
\texttt{\textquoteright}
\texttt{\textquoteright}}
\providecommand{\grqq}{% 
\texttt{\textquoteright}
\texttt{\textquoteright}}

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

The French double guillemets.

7.12.4 Umlauts and tremas

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

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

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.

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

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

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

\ifx\l@english\@undefined\chardef\l@english\@undefined\fi
\ifx\l@unhyphenated\@undefined\newlanguage\l@unhyphenated\fi

7.13 Layout

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

\bbl@trace{Input engine specific macros}
\ifcase\bbl@engine
\input txtbabel.def
\or
\input luababel.def
\or
\input xebabel.def
\fi

7.15 Creating and modifying languages

\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.

\bbl@trace{Creating languages and reading ini files}
\let\bbl@extend@ini\@gobble
\newcommand\babelprovide[2][\@nil]{% 
\let\bbl@savelangname\languagename
\edef\bbl@savelocaleid{\the\localeid}%
% Set name and locale id
\edef\languagename{#2}%
\bbl@id@assign
% Initialize keys
\let\bbl@KVP@captions\@nil
\let\bbl@KVP@date\@nil
\let\bbl@KVP@import\@nil
\let\bbl@KVP@main\@nil
\let\bbl@KVP@script\@nil
\let\bbl@KVP@language\@nil
\let\bbl@KVP@hyphenrules\@nil
\let\bbl@KVP@linebreaking\@nil
\let\bbl@KVP@justification\@nil
\let\bbl@KVP@mapfont\@nil
\let\bbl@KVP@maparabic\@nil
\let\bbl@KVP@mapdigits\@nil
\let\bbl@KVP@intraspace\@nil
\let\bbl@KVP@intrapenalty\@nil
\let\bbl@KVP@onchar\@nil
\let\bbl@KVP@transforms\@nil
\global\let\bbl@release@transforms\@empty
\global\let\bbl@extend@ini\@gobble
\gdef\bbl@key@list{;}%
\bbl@forkv{#1}{% TODO - error handling
\in@{/}{##1}%
\ifin@
\global\let\bbl@extend@ini\bbl@extend@ini@aux
\bbl@renewinikey##1\@@{##2}%
\else
\bbl@csarg\def{KVP@##1}{##2}%
\fi}%
\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\@nil\else
  \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}\fi
\ifx\bbl@KVP@language\@nil\else
  \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}\fi
% == onchar ==
\ifx\bbl@KVP@onchar\@nil\else
  \bbl@luahyphenate
  \bbl@exp{\AddToHook{env/document/before}{\select@language{#2}{}}}%
\directlua{
if Babel.locale_mapped == nil then
  Babel.locale_mapped = true
  Babel.linebreaking.add_before(Babel.locale_map)
  Babel.loc_to_scr = {} 
  Babel.chr_to_loc = Babel.chr_to_loc or {}
end}%
\bbl@xin{ ids }{ \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}']
  Babel.locale_props[\the\localeid].lc = \the\localeid\space
  Babel.locale_props[\the\localeid].lg = \the@nameuse{l@\languagename}\space
end
}%
\bbl@xin{ fonts }{ \bbl@KVP@onchar\space}%
\ifin@
  \bbl@imapselect\undefined % TODO. almost the same as mapfont
  \AtBeginDocument{%
    \bbl@patchfont{\bbl@mapselect}{\selectfont}{\bbl@mapselect}{%}
    \def\bbl@mapselect{%
      \let\bbl@mapselect\relax
\endinput
\ifunset{bbl@dgnat@languagename}{}%\RequirePackage{luatexbase}\bbl@activate@preotf\directlua{Babel = Babel or {} %%% -> presets in luababelBabel.digits_mapped = trueBabel.digits = Babel.digits or {}Babel.digits[\the\localeid] = table.pack(string.utfvalue('bbl@cl{dgnat}'))if not Babel.numbers thenfunction Babel.numbers(head)local LOCALE = Babel.attr_localelocal GLYPH = node.id'glyph'local inmath = falsefor item in node.traverse(head) doif not inmath and item.id == GLYPH thenlocal temp = node.get_attribute(item, LOCALE)if Babel.digits[temp] thenlocal chr = item.charif chr > 47 and chr < 58 thenitem.char = Babel.digits[temp][chr-47]endendelseif item.id == node.id'math' theninmath = (item.subtype == 0)endendreturn headendend}}%\fi\fi% == Counters: alph, Alph ==% What if extras<lang> contains a \bbl@save@alph? It won’t be% restored correctly when exiting the language, so we ignore% this change with the \bbl@alph@saved trick.%\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\fi\fi\ifx\bbl@KVP@alph\@nil\else\bbl@extras@wrap{\bbl@alph@saved}\{\let\bbl@alph@saved\@alph\let\@alph\bbl@alph@saved\babel@save\@alph\bbl@exp{\bbl@add<extras\languagename>{\let\@alph<\bbl@cntr@\bbl@KVP@alph @\languagename}}\bbl@add\{\bbl@KVP@Alph\@nil\else\bbl@add\{\bbl@KVP@Alph\@nil\else\bbl@add\{\bbl@KVP@Alph\@nil\else\bbl@add\{\let\@alph<\bbl@cntr@\bbl@KVP@Alph @\languagename}\}}}%\fi\fi\fi\ifx\bbl@KVP@Alph\@nil\else\bbl@add\{\bbl@KVP@Alph\@nil\else\bbl@add\{\bbl@KVP@Alph\@nil\else\bbl@add\{\let\@alph<\bbl@cntr@\bbl@KVP@Alph @\languagename}\}}%\fi\fi% == require.babel in ini ==% To load or reaload the babel-*\text, if require.babel in ini% \text{\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\fi}}%}}}
Depending on whether or not the language exists (based on \date<language>), we define two macros. Remember \bbl@startcommands opens a group.

\begin{verbatim}
\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\@nil % and also if import, implicit
  \def\bbl@tempb##1{% elt for \bbl@captionslist
    \bbl@exp{\SetString\##1{\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
    \expandafter\bbl@tempb
  }
  \expandafter\bbl@tempb\bbl@captionslist\@empty
\else
  \ifx\bbl@initoload\relax
    \bbl@read@ini\bbl@initoload2% % Here letters cat = 11
  \else
    \bbl@read@ini\bbl@initoload2% % Same
  \fi
  \bbl@savetoday
  \bbl@savetoday
\end{verbatim}
Load the basic parameters (ids, typography, counters, and a few more), while captions and dates are left out. But it may happen some data has been loaded before automatically, so we first discard the saved values. (TODO. But preserving previous values would be useful.)

The hyphenrules option is handled with an auxiliary macro.
\\@exp{% and hyphenrules is not empty
  \ifblank{\cs{hyphr}@#1}{%\fi
  \let\tempa<\l@\cl{hyphr}>}}% 
\fi
\else % if importing
  \ifblank{\cs{hyphr}@#1}{%\fi
  \let\tempa<\l@\cl{hyphr}>}}% 
\fi
\ifunset{\tempa}% ie, relax or undefined
  \ifunset{\l@#1}% no hyphenrules found - fallback
    \exp{\adddialect<\l@#1>\language}}% \else, \l@<lang> is ok - nothing to do
  \exp{\adddialect<\l@#1>\tempa}}% found in opt list or ini
\fi
\def\input@texini#1{%\bbl@bsphack\exp{%\catcode`\\=0 \catcode`\%=14 \catcode`\{=1 \catcode`\}=2\lowercase{\InputIfFileExists{babel-#1.tex}{}{}}\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\iniline#1\iniline{%\@ifnextchar[\inisect{\@ifnextchar;\iniskip\instore}#1\@@} \def\inisect[#1]#2\@@{%\def\section{#1}} \def\iniskip#1\@@{% if starts with ; \def\instore#1=#2\@@{% full (default)
  \trim{\def\tempa{#1}}\trim{\toks{#2}}\xin{.identification.}{.\section.}\ifin\else\exp{\g@addto@macro\inidata{\section}{\tempa}{\toks}}\fi}
\def\instore@min#1=#2\@@{% minimal (maybe set in \bbl@read@ini)
  \trim{\def\tempa{#1}}\trim{\toks{#2}}\xin{.identification.}{.\section.}\ifin\else\exp{\g@addto@macro\inidata{\section}{\tempa}{\toks}}\fi}
Now, the 'main loop', which **must be executed inside a group**. At this point, \inidata may contain data declared in \provide, with 'slashed' keys. There are 3 steps: first read the ini file and store it; then traverse the stored values, and process some groups if required (date, captions, labels, counters); finally, 'export' some values by defining global macros (identification, typography, characters, numbers). The second argument is 0 when called to read the minimal data for fonts; with \provide it's either 1 or 2.
\ifx\readstream\undefined
  \csname newread\endcsname\bbl@readstream
\else

\end{document}
\def\bbl@readini#1{% \
\global\let\bbl@extendini@\@gobble 
\openin\bbl@readstream=babel-#1.ini
\ifeof\bbl@readstream
\bbl@error
{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.}%
\else
% == Store ini data in \bbl@inidata == 
\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\@@ 
\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 
% == Process stored data == 
\bbl@csarg\xdef{lini@\languagename}{#1}% 
\bbl@readini@aux 
% == 'Export' data == 
\bbl@export\clessthan\bbl@inidata\clessthan
\bbl@saveini@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@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@extendini@\@gobble 
\bbl@startcommands*{#1}{captions}% 
% Activate captions/... and modify exports
As somewhat hackish tool to handle calendar sections. To be improved.

The previous assignments are local, so we need to export them. If the value is empty, we can provide default value.
Key-value pairs are treated differently depending on the section in the ini file. The following macros are the readers for identification and typography. Note `\bbl@ini@exports` is called always (via `\bbl@inisec`), while `\bbl@after@ini` must be called explicitly after `\bbl@read@ini` if necessary.
A shared handler for `key=val` lines to be stored in `\bbl@kv@<section>..<key>`.}

\def\bbl@inikv#1#2{%
  \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@typography\bbl@inikv
\let\bbl@inikv@characters\bbl@inikv
\let\bbl@inikv@numbers\bbl@inikv

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

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

\def\bbl@ini@captions@template#1#2{% string \language tempa=\caption{\texttt{name}.}}%
\def\bbl@replace\bbl@tempa{\template}{%%
\bbl@replace\bbl@tempa{[ ]}{\nobreakspace{}}%
\bbl@replace\bbl@tempa{[ ]}{\csname}%
\bbl@replace\bbl@tempa{[ ]}{\csname the}%
\bbl@replace\bbl@tempa{[ ]}{\name\endcsname{}}%
\bbl@replace\bbl@tempa{[ ]}{\endcsname{}}%
\bbl@xin@{\bbl@tempa}{\chapter,appendix,part,}\%}
\@nameuse{\bbl@patch\bbl@tempa}%
\global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
Labels. Captions must contain just strings, no format at all, so there is new group in ini files.
The following code is still under study. You can test it and make suggestions. Eg, enumerate.2 = ([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\endcsname{}}}\
\toks@\expandafter{\bbl@toreplace}\
\% TODO. Execute only once:
\bbl@exp{\bbl@add\appendix{\languagename}{}\
\bbl@ifunset{bbl@chaptype fmt@\languagename}{}\
{\@chapapp\space\thechapter}{\@nameuse{bbl@chaptype fmt@\languagename}}}\
\bbl@ifunset{bbl@partfmt@\languagename}{}\
{\partname\nobreakspace{\thepart}}{\@nameuse{bbl@partfmt@\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 strings, while in Hungarian it 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@bl@chaptype fmt@\languagename}{}\bbl@ifunset{bbl@partfmt@\languagename}{\partname\nobreakspace{\thepart}}{\@nameuse{bbl@partfmt@\languagename}}}\fi
\fi
\fi
\bbl@sreplace\@part{\partname\nobreakspace{\thepart}}{\bbl@chfmt}\
\bbl@sreplace\chaptermark{\@chapapp\space\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\if\fi
\else\def\bbl@patchchapter{%\
\global\let\bbl@patchchapter\relax
\gdef\bbl@chfmt{%\
\bbl@ifunset{bbl@chaptype fmt@\languagename}{\@chapapp\space\thechapter}
\gdef\bbl@ifunset{bbl@chaptype fmt@\languagename}{}\
{\@nameuse{bbl@chaptype fmt@\languagename}}}\fi
\fi\fi
\fi
\else\def\bbl@patchchapter{%\
\global\let\bbl@patchchapter\relax
\gdef\bbl@chaptypefmt{%\
\bbl@ifunset{bbl@chaptype fmt@\languagename}{\@chapapp\space\thechapter}
\gdef\bbl@ifunset{bbl@chaptype fmt@\languagename}{}\
{\@nameuse{bbl@chaptype fmt@\languagename}}}\fi
\fi
\fi
\let\bbl@calendar\@empty
\DeclareRobustCommand\localedate[1][]{\bbl@localedate{#1}}\let\bbl@calendar\@empty
\def\bbl@localedate#1#2#3#4{\begingroup\ifx\@empty#1\@empty\else
\let\bbl@ld@calendar\@empty\let\bbl@ld@variant\@empty\edef\bbl@tempa{%\zap@space#1 \@empty}\def\bbl@tempb##1=##2\@@{%\@namedef{bbl@ld@##1}{##2}}\bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}\edef\bbl@calendar{\bbl@ld@calendar\ifx\bbl@ld@variant\@empty\else.bl@ld@variant\fi}\bbl@replace\bbl@calendar{gregorian}{}\fi\bbl@cased\{\@nameuse{bbl@date@\languagename @\bbl@calendar}{#2}{#3}{#4}\}\endgroup}
\def\bbl@inidate#1.#2.#3.#4\relax#5#6{%\bbl@trim@def\bbl@tempa{#1.#2}\bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate\{\bbl@trim{\bbl@tempa}{#3}\}\bbl@trim{\toks@}{#5}\bbl@exp{\@temptokena\expandafter{\@nameuse{\bbl@savedate}}}% Reverse order - in ini last wins\bbl@exp{\bbl@tempb{#6}}% defined now\bbl@trim{\toks@}{#5}\bbl@TG@@date\bbl@ifunset{bbl@date@\languagename @}% TODO. Move to a better place.\{\@nameuse{bbl@date@\languagename date}@@protect\}@\language}{\\\bbl@usedategrouptrue\\\edef\language{####1####2####3}{%\\bbl@add\\bbl@savetoday{\language}{\the\year}{\the\month}{\the\day}}}\global\bbl@csarg\let{date@\languagename @}{\bbl@toreplace}\global\bbl@csarg\let{date@\languagename @\bbl@tempb}{\bbl@toreplace}}\global\bbl@csarg\let{date@\languagename @}{\bbl@toreplace}\global\bbl@csarg\let{date@\languagename @\bbl@tempb}{\bbl@toreplace}\ifx\bbl@tempb\@empty\else\global\bbl@csarg\let{date@\languagename @\bbl@tempb}{\bbl@toreplace}\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 either in the date or in the month name. Note after \bbl@replace \toks@ contains the resulting string, which is used by \bbl@replace@finish@iii (this implicit behavior doesn’t seem a good idea, but it’s efficient).
\newcommand\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]{{\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{Currently two-digit years are restricted to the\range 0-9999.}There is little you can do. Sorry.\}%}
\newcommand\BabelDateyyyy[1]{{\number#1}} % TODO - add leading 0
\def\bbl@replace@finish@iii#1{\bbl@exp{\def\#1####1####2####3{\the\toks@}}}
\def\bbl@ TG@@date{\bbl@replace\bbl@toreplace{[\ ]}{\BabelDateSpace{}}\bbl@replace\bbl@toreplace{[.]}{.\@}\bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}\bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}\bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}\bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}\bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}\bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}\bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}\bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}\bbl@replace\bbl@toreplace{[y|}{\bbl@datecntr{#1|}}\bbl@replace\bbl@toreplace{[m|}{\bbl@datecntr{#2|}}\bbl@replace\bbl@toreplace{[d|}{\bbl@datecntr{#3|}}\bbl@replace@finish@iii\bbl@toreplace}
\def\bbl@datecntr{\expandafter\bbl@xdatecntr\expandafter}
\def\bbl@xdatecntr[#1|#2]{\localenumeral{#2}{#1}}

Transforms.
\let\bbl@release@transforms\@empty
\namedef\bbl@ inikv@transforms.prehyphenation{%\bbl@transforms\babelprehyphenation}
\namedef\bbl@ inikv@transforms.posthyphenation{%\bbl@transforms\babelposthyphenation}
\def\bbl@transforms@aux#1#2#3,#4,#5\relax{#1[#2]{#3}{#4}{#5}}
\begingroup % A hack. TODO. Don't require an specific order
\catcode`%=12
\catcode`&=14
\gdef\bbl@transforms#1#2#3{&%\ifx\bbl@KVP@transforms\@nil\else\directlua{local str = "==[#2]=="\str = str:gsub\("%.%.d+.%.d+$",\"")\tex.print("[[\def\string\babeltempa[] .. \str .. \{\}]]")\}%\bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%\ifin@
in@{.0$}{#2$}&%\ifie\bbl@release@transforms\@empty\else\bbl@transforms\bbl@prehyphenation\et\ifie\bbl@release@transforms\@empty\else\bbl@posthyphenation\et\ifie\begingroup % A hack. TODO. Don't require an specific order
\catcode`\%=12
\catcode`\&=14
\gdef\bbl@transforms\#1\#2\#3\&%\ifx\bbl@KVP@transforms\@nil\else\directlua{local str = [[\#2]]\str = str:gsub\("%.%.d+.%.d+$",\"")\tex.print\("[[\def\string\babeltempa[] .. \str .. \{\}]]")\}%\bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%\ifie\bbl@release@transforms\@empty\else\bbl@transforms\bbl@prehyphenation\et\ifie\bbl@release@transforms\@empty\else\bbl@posthyphenation\et\ifie

127
Language and Script values to be used when defining a font or setting the direction are set with the following macros.

`\def\bbl@provide@lsys#1{\bbl@ifunset{bbl@lname@#1}{\bbl@load@info{#1}}{}}\bbl@csarg\let{lsys@#1}@empty\bbl@csarg\gdef{sname@#1}{Default}\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}{}{\let\bbl@xenohyph\bbl@xenohyph@d\ifx\AtBeginDocument\@notprerr\expandafter\@secondoftwo % to execute right now\expandafter\selectlanguage\expandafter{\languagename}\fi}}% \bbl@csarg\bbl@toglobal{lsys@#1}%%
`
The following ini reader ignores everything but the identification section. It is called when a font is defined (i.e., when the language is first selected) to know which script/language must be enabled. This means we must make sure a few characters are not active. The ini is not read directly, but with a proxy .tex file named as the language (which means any code in it must be skipped, too).

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

The first macro is the generic “localized” command.

Alphabetic counters must be converted from a space separated list to an \ifcase structure.
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@languageaname}{#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{\bbl@ifunset{bbl@cntr@#1.F.\number#2@languageaname}{}{\bbl@cs{cntr@#1.F.\number#2@languageaname}}}
\def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8\@@{\bbl@ifunset{bbl@cntr@#1.4@languageaname}{}{\bbl@cs{cntr@#1.4@languageaname}}}
\def\bbl@alphnum@invalid#1{\bbl@error{Alphabetic numeral too large (#1)}{Currently this is the limit.}}

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

\newcommand\localeinfo[1]{\ifx\InputIfFileExists\@undefined\else\bbl@ensureinfo##1{\bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}\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}[1][]{% \@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 key for locale '#2':\% #3\% \string#1 will be set to \relax}% {Perhaps you misspelled it.}% \fi}\let\bbl@ini@loaded@empty\newcommand{\LocaleForEach}{\bbl@foreach\bbl@ini@loaded}

8 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}{% \@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 % The error is gobbled if everything went ok. {Currently, #1 related features can be adjusted only\% in the main vertical list.}% {Maybe things change in the future, but this is what it is.}}% \namedef{bbl@ADJ@bidi.mirroring@on}{\bbl@adjust@lua{bidi}{mirroring_enabled=true}}\namedef{bbl@ADJ@bidi.mirroring@off}{\bbl@adjust@lua{bidi}{mirroring_enabled=false}}\namedef{bbl@ADJ@bidi.text@on}{\bbl@adjust@lua{bidi}{bidi_enabled=true}}\namedef{bbl@ADJ@bidi.text@off}{\bbl@adjust@lua{bidi}{bidi_enabled=false}}\namedef{bbl@ADJ@bidi.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.sea@off}{% \
bbl@adjjust@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% The error is gobbled if everything went ok.
  \{Currently, layout related features can be adjusted only\%
  \ in vertical mode.\}% 
  \{Maybe things change in the future, but this is what it is.\})% 
  \namedef{bbl@ADJ@layout.tabular@on}{% 
  \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}% 
  \namedef{bbl@ADJ@layout.tabular@off}{% 
  \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}% 
  \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@hyphenation.extra@on}{% 
  \bbl@activateposthyphen}% 
  \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@select.write@shift}{% 
    \let\bbl@restorelastskip\relax% 
    \def\bbl@savelastskip{% 
      \let\bbl@restorelastskip\relax% 
      \ifvmode% 
        \ifdim\lastskip=\z@% 
          \endgraf
As the final task, load the code for lua. TODO: use babel name, override

As the final task, load the code for lua. TODO: use babel name, override

As the final task, load the code for lua. TODO: use babel name, override

The \LaTeX{} book states:

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

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

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

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

\begin{itemize}
\item\texttt{safe=none}: \let\bbl@opt@safe\@empty
\item\texttt{safe=bib}: \def\bbl@opt@safe{B}
\item\texttt{safe=ref}: \def\bbl@opt@safe{R}
\end{itemize}

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

\begin{example}
\end{example}

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.

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

\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}\
      \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}\
      \ifx\bbl@tempa\bbl@tempb
        \else
          \@tempswatrue
      \fi

  \fi
}

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

\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.

\def\@citex[#1][#2]#3{\@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
  \org@@citex[#1][#2]{\@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}{%
    \def\@citex[#1][#2][#3]{\@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
      \org@@citex[#1][#2][#3]{\@tempa}}%
}}
The package cite has a definition of \@citex where the shorthands need to be turned off in both arguments.

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

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

```latex
\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@activetrue is in effect. This switch needs to be reset inside the \hbox which contains the citation label. In order to determine during .aux file processing which definition of \bibcite is needed we define \bibcite in such a way that it redefines itself with the proper definition. We call \bbl@cite@choice to select the proper definition for \bibcite. This new definition is then activated.

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

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

```latex
\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.

```latex
\AtBeginDocument{\bbl@cite@choice}
```

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

```latex
\def\@bibitem#1{%
  \@safe@activetrue
  \org@bibitem{#1}\@safe@activesfalse}
```

8.2 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 hidi documents.

```latex
\bbl@trace{Marks}
```

```latex
\IfBabelLayout{sectioning}{\iffalse\bbl@opt@headfoot\nil
\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}\i}
\ifbbl@single\else
\bbl@ifunset{markright }\bbl@redefine\bbl@redefinerobust
\markright#1{%
\bbl@ifblank{#1}{}
{\toks@{#1}\bl@exp{\\\org@markright{\\\protect\\\foreignlanguage{\languagename}\\\protect\\\bbl@restore@actives\the\toks@}}}}%
\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

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

\@mkboth

8.3 Preventing clashes with other packages

8.3.1 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.

8.3.2 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.

8.3.3 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.
\AtBeginDocument{%
\ifpackageloaded{hhline}{}
\else
  \makeatletter
  \def\@currname{hhline}\input{hhline.sty}\makeatother
\fi%
}}

\substitutefontfamily

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

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

8.4 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,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
\AtBeginDocument{%
  \def\@elt#1{,#1,}\
  \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}\
  \let\@elt\relax\let\bbl@tempb\@empty\let\bbl@tempc\@empty
  \bbl@foreach{\BabelNonASCII}{{LGR loaded in a non-standard way}}\
  \bbl@foreach{\bbl@tempa}{{\ifin@\def\bbl@tempb{#1}% Store last non-ascii\else\bbl@tempb\fi}}\
  \bbl@foreach{\bbl@tempb}{{\ifin@\def\bbl@tempc{#1}% Store last ascii\else\bbl@tempc\fi}}\
}\@onlypreamble\substitutefontfamily

138
\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.

\latinencoding AtEndOfPackage\{\edef\latinencoding{\cfencoding}\}\par
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.

\latinencoding AtBeginDocument\{%\@ifpackageloaded{fontspec}%\{\xdef\latinencoding{%\ifx\UTFencname\@undefined\EU\ifcase\bbl@engine\or2\or1\fi\else\UTFencname\fi}\}\par
\latinencoding But for several functions, we need to execute some code with \selectfont. With \TeX{} 2021-06-01, there is a hook for this purpose, but in older versions the \TeX{} command is patched (the latter solution will be eventually removed).

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

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

\textlatin For several functions, we need to execute some code with \selectfont. With \TeX{} 2021-06-01, there is a hook for this purpose, but in older versions the \TeX{} command is patched (the latter solution will be eventually removed).
8.5 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 groupings.
• luatex can provide the most complete solution, as we can manipulate almost freely the node list,
the generated lines, and so on, but bidi text does not work out of the box and some development
is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As
Lua\TeX-ja shows, vertical typesetting is possible, too.

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

\def\bbl@alscripts{,Arabic,Syriac,Thai,}
\def\bbl@rscripts{% TODO. Base on codes ??
,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaean,%
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@ % useless in xetex
\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{% TODO - math
\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 - strictly, not the right test
\bbl@bodydir{#1}\bbl@pardir{#1} %
\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\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 - not the right test
\bbl@bodydir{\#1} %
\bbl@pardir{\#1} %
\fi}
\\bbl@textdir{\#1} %
\% TODO. Only if \bbl@bidimode > 0:
\bbl@exp{\\bbl@switchdir}\bbl@textdir{\#1} %
\% TODO. Must be moved to the engine files.
The following command is executed only if there is a right-to-left script (once). It activates the \everypar hack for xetex, to properly handle the par direction. Note text and par dirs are decoupled to some extent (although not completely).

\def\bbl@xeverypar{%}
\let\bbl@xeverypar\relax
\TeXeTstate\@ne
\def\bbl@xeverypar{%}
\ifcase\bbl@thepardir\relax
\fi
\let\bbl@severypar\everypar
\let\bbl@severypar\relax
\let\bbl@severypar\@empty
\AddBabelHook{bidi}{foreign}{%}
\def\bbl@tempa{%}
\def\BabelText####1}{

142
A tool for weak L (mainly digits). We also disable warnings with hyperref.

8.6 Local Language Configuration

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

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

8.7 Language options

Languages are loaded when processing the corresponding option except if a main language has been set. In such a case, it is not loaded until all options has been processed. The following macro inputs the ldf file and does some additional checks (\input works, too, but possible errors are not caught).
Now, we set a few language options whose names are different from .ldf files. These declarations are preserved for backwards compatibility, but they must be eventually removed. Use proxy files instead.

\def\bbl@try@load@lang#1#2#3{\
  \IfFileExists{\CurrentOption.ldf}\
    {\bbl@load@language{\CurrentOption}}\
  {#1\bbl@load@language{#2}{#3}}}

\DeclareOption{hebrew}{\input{rl babel.def}\
  \bbl@load@language{hebrew}}
\DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
\DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
\DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
\DeclareOption{polutonikogreek}{\bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}\n\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%\
  *})%\n  {}%\n}{%\n  \else\n  \InputIfFileExists{\bbl@opt@config.cfg}%\n  \typeout{****************************************\^J%\
  * Local config file \bbl@opt@config.cfg used\^J%\
  *)%\n  {}%\n  \fi\n
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.

\ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
  \let\bbl@temph\@empty\n  \edef\bbl@tempb{\classoptionslist,\bbl@language@opts}\n  \bbl@foreach\bbl@tempb{#1,\bbl@tempb}\n  {\edef\bbl@tempb{#1,\bbl@tempb}}%\n  \bbl@foreach\bbl@tempb{% #1 is a reversed list\n    \edef\bbl@tempb{#1,\bbl@tempb}%\n    \ifnum\bbl@iniflag=\z@% if not yet assigned\n      \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}\n      \else% n +=\n        \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}\n      \fi%\n    \fi%\n  }%\n  \else\n    \bbl@foreach\bbl@tempb{%\n      \bbl@error{% Local config file \bbl@opt@config.cfg' not found}%\n      \Perhaps you misspelled it.}%\n  \fi\n  \fi%\n\else
A few languages are still defined explicitly. They are stored in case they are needed in the 'main' pass (the value can be \relax).

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

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):

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.
\edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
\let\bbl@tempc\@empty
\bbl@for\bbl@tempb\bbl@tempa{\bbl@xin{,\bbl@tempb,}{,\bbl@loaded,}\
\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{\textbf{Last declared language option is '}, \texttt{\bbl@tempc}',\textbackslash
\textbackslash
but the last processed one was '}, \texttt{\bbl@tempb}'.\textbackslash
The main language can't be set as both a global\textbackslash
and a package option. Use 'main=', \texttt{\bbl@tempc} as\textbackslash
option. Reported}
\fi
\else
\ifodd\bbl@iniflag % case 1,3 (main is ini)
\bbl@ldfinit
\let\CurrentOption\bbl@opt@main
\bbl@exp{{\textbackslash}\bbl@opt@provide = empty if *}\
{\textbackslash}\bbl@provide{[\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}
\ExecuteOptions{\bbl@opt@main}
\@namedef{ds@\bbl@opt@main}{}%
\fi
\DeclareOption*{}
\ProcessOptions*
\fi
\def\AfterBabelLanguage{%
\bbl@error{\textbf{Too late for \string\AfterBabelLanguage}}%
{\textbf{Languages have been loaded, so I can do nothing}}}

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.
\edef\bbl@main@language@undefined\edef\bbl@info{%
You haven't specified a language. I'll use 'nil'\textbackslash
as the main language. Reported}
\edef\bbl@load@language(nil)\fi
\}(/package)

9 The kernel of Babel (babel.def, common)

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

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

\let\bbl@onlyswitch\@empty
\input babel.def
\let\bbl@onlyswitch\@undefined
\let\@empty{}
\ProvidesFile{hyphen.cfg}[date version] Babel hyphens
\xdef\jobname{}
\def\version{
\version}
\def\date{
\date}
\ifx\AtBeginDocument\@undefined
\fi
\Definecoreswitchingmacros\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.
\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.
\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.

\process@line
\process@synonym
\process@language

10 Loading hyphenation patterns

The following code is meant to be read by \input \TeX{} because it should instruct \TeX{} to read hyphenation patterns. To this end the docstrip option patterns is used to include this code in the file hyphen.cfg. Code is written with lower level macros.
For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file language.dat by adding for instance ‘:T1’ to the name of the language. The macro \bbl@get@enc extracts the font encoding from the language name and stores it in \bbl@hyph@enc. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

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

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

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

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

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

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

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

\def\bbl@hook@everylanguage#1{\relax}
\def\bbl@hook@loadpatterns#1{\input #1\relax}
\def\bbl@hook@loadexceptions\bbl@hook@loadpatterns
\def\addlanguage{\csname newlanguage\endcsname}\relax
\def\adddictionary#1#2\relax
\wlog{\string#1 = a dialect from \string#2}\relax
\def\iflanguage#1{\expandafter\ifx\csname l@#1\endcsname\relax
\else\expandafter\expandafter\expandafter\@firstoftwo
\else\expandafter\expandafter\expandafter\@secondoftwo\fi}
\def\providehyphenmins#1#2{\expandafter\ifx\csname #1hyphenmins\endcsname\relax
\@namedef{#1hyphenmins}{#2}\fi}
\def\set@hyphenmins#1#2{\lefthyphenmin#1\relax
\righthyphenmin#2\relax}
\def\selectlanguage{\errhelp{Selecting a language requires a package supporting it}\
\errmessage{Not loaded}}
\let\foreignlanguage\selectlanguage
\let\otherlanguage\selectlanguage
\let\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}
\input xebabel.def
\ifx\directlua\@undefined\else\input xebabel.def\fi
\input babel-bl@format.cfg
\ifx\directlua\@undefined\else\input xebabel.def\fi
\openin1=babel-bl@format.cfg
\ifeof1\else\input babel-bl@format.cfg\relax\fi
\fi
\readconfigfile The configuration file can now be opened for reading.
\openin1 = language.dat
See if the file exists, if not, use the default hyphenation file hyphen.tex. The user will be informed about this.
\def\languagename{english}\
\ifeof1
    \message{I couldn't find the file language.dat, I will try the file hyphen.tex}
    \input hyphen.tex\relax
    \chardef\l@english\z@
\else
    Pattern registers are allocated using count register \last@language. Its initial value is 0. The definition of the macro \newlanguage is such that it first increments the count register and then defines the language. In order to have the first patterns loaded in pattern register number 0 we initialize \last@language with the value −1.
\last@language\m@ne
We now read lines from the file until the end is found. While reading from the input, it is useful to switch off recognition of the end-of-line character. This saves us stripping off spaces from the contents of the control sequence.
\loop
\endlinechar\m@ne
\read1 to \bbl@line
\endlinechar`\^^M
If the file has reached its end, exit from the loop here. If not, empty lines are skipped. Add 3 space characters to the end of \bbl@line. This is needed to be able to recognize the arguments of \process@line later on. The default language should be the very first one.
\if T\ifeof1F\fi T\relax
\ifx\bbl@line\@empty\else
    \edef\bbl@line{\bbl@line\space\space\space}\
    \expandafter\process@line\bbl@line\relax
\fi
\repeat
Check for the end of the file. We must reverse the test for \ifeof without \else. Then reactivate the default patterns, and close the configuration file.
\begingroup
\def\bbl@elt#1#2#3#4{\global\language=#2\relax
\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
Font handling with fontspec

Add the bidi handler just before luaotload, 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.

\begin{verbatim}
\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
\end{verbatim}

Here the code for \texttt{inI\_T\_E\_X} ends.

11 Font handling with fontspec

Add the bidi handler just before luaotload, 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.

\begin{verbatim}
\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
\end{verbatim}

\section{Font handling with fontspec}

Add the bidi handler just before luaotload, 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.

\begin{verbatim}
\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
\end{verbatim}

Here the code for \texttt{inI\_T\_E\_X} ends.

11 Font handling with fontspec

Add the bidi handler just before luaotload, 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.

\begin{verbatim}
\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
\end{verbatim}

Here the code for \texttt{inI\_T\_E\_X} ends.

11 Font handling with fontspec

Add the bidi handler just before luaotload, 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.

\begin{verbatim}
\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
\end{verbatim}

Here the code for \texttt{inI\_T\_E\_X} ends.
If the family in the previous command does not exist, it must be defined. Here is how:
\begin{verbatim}
\def\bbl@providefam#1{% 
\expandafter\bbl@ifblank\expandafter{\bbl@tempa}{% save bbl@rmdflt@ 
\bbl@exp{\let\<bbl@##1dflt@\languagename>\<bbl@##1dflt@>}{% (3) from generic 
\global\let\<bbl@##1dflt@\languagename>\<bbl@##1dflt@>}}% (2) from script 
\bbl@foreach\bbl@font@fams{% (1) language? 
\bbl@exp{\let\<bbl@##1dflt@\languagename>\<bbl@##1dflt@>}{% (2) from script? 
\bbl@exp{\let\<bbl@##1dflt@\languagename>\<bbl@##1dflt@>}{% (3) from generic? 
\global\let\<bbl@##1dflt@\languagename>\<bbl@##1dflt@>}}%}
\end{verbatim}

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.
\begin{verbatim}
\def\bbl@nostdfont#1{% 
\bbl@ifunset{bbl@WFF\f@family}{% Flag, to avoid dupl warns 
\bbl@csarg\gdef{WFF\f@family}{}% Flag, to avoid dupl warns 
\bbl@infowarn{The current font is not a babel standard family:\% 
\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 no 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})}
\end{verbatim}

\begin{verbatim}
\def\gdef\bbl@switchfont{% 
\lowercase{\edef\<\bbl@tempa\languagename>\<\bbl@tempa\languagename>}}% 
\bbl@foreach\bbl@font@fams{% 
\bbl@exp{% 
\\bbl@ifblank\expandafter{\bbl@##1dflt@\languagename}@\languagename}{% (1) language? 
\bbl@exp{% 
\\bbl@ifblank\expandafter{\bbl@##1dflt@\languagename}@\languagename}{% (2) from script? 
\bbl@exp{% 
\\bbl@ifblank\expandafter{\bbl@##1dflt@\languagename}@\languagename}{% (3) from generic? 
\global\let\<bbl@##1dflt@\languagename>\<bbl@##1dflt@>}}%}
\end{verbatim}

\begin{verbatim}
\def\bbl@loadfonspec{\edef\bbl@tempa{#1}% 
\def\bbl@tempb{#2}% Used by \bbl@bblfont 
\ifx\fontspec\@undefined{\bbl@loadfontspec}% Just calls \bbl@switchfont 
\else
\EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont 
\fi 
\newcommand\bbl@bblfont[#2]{% 1=features 2=fontname, @font=rm|sf|tt 
\bbl@ifunset{\bbl@tempb family}{% For the default font, just in case: 
\bbl@providefam{\bbl@tempb}{% 1=features 2=fontname, @font=rm|sf|tt 
\bbl@exp{% 
\let\<bbl@##1dflt@\languagename>\<bbl@##1dflt@>\bbl@font@set\<bbl@##1dflt@\languagename>\family\<bbl@##1dflt@}\default\family}{% (3) from generic 
\global\let\<bbl@##1dflt@\languagename>\<bbl@##1dflt@>}}% (2) from script 
\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt 
\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
\bbl@exp{% 
\let\<bbl@##1dflt@\languagename>\<bbl@##1dflt@>\bbl@font@set\<bbl@##1dflt@\languagename>\family\<bbl@##1dflt@}\default\family}{% (3) from generic 
\global\let\<bbl@##1dflt@\languagename>\<bbl@##1dflt@>}}% (2) from script 
\bbl@exp{% 
\\bbl@ifblank\expandafter{\bbl@tempb}{% which sets a flag to avoid duplicate them. 
\bbl@ifunset{bbl@WFF\f@family}{% Flag, to avoid dupl warns 
\bbl@csarg\gdef{WFF\f@family}{}% Flag, to avoid dupl warns 
\bbl@infowarn{The current font is not a babel standard family:\% 
\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 no 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})}
\end{verbatim}
The following is executed at the beginning of the aux file or the document to warn about fonts not defined with `\babelfont`.

Now the macros defining the font with `fontspec`.

When there are repeated keys in `fontspec`, the last value wins. So, we just place the ini settings at the beginning, and user settings will take precedence. We must deactivate temporarily `\bbl@mapselect` because `\selectfont` is called internally when a font is defined.
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@fmdflt@lang fnt-opt fnt-mne \xxfamily
\let\bbl@tempe\bbl@mapselect
\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{soff}}% 
\<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}% 
\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}% 
\renewfontfamily{\#4}%
[
\bbl@cl{lsys},#2]\#3}% ie \bbl@exp{..}{#3}
\begingroup
#4%
\edef#1\{\f@family% eg, \bbl@fmdflt@lang{FreeSerif(0)}
\endgroup
\let#4\bbl@temp@fam
\bbl@exp{\let\\bbl@stripslash#4\space}{\bbl@temp@pfam}
\let\bbl@mapselect\bbl@tempe}

\def\bbl@font@rst#1#2#3#4{%
\def\famrst@#4{\bbl@font@set{#1}#2#3}
% default font families. They are eurocentric, but the list can be expanded easily with \babelfont.
% The old tentative way. Short and preserved for compatibility, but deprecated. Note there is no direct alternative for \babelfont\#features. The reason in explained in the user guide, but essentially – that was not the way to go :-).
%\newcommand\babelfSstore[2][]{%
% \bbl@ifblank{#1}%
% {\bbl@csarg\def{\sname@#2}{Latin}}%
% {\bbl@csarg\def{\sname@#2}{#1}}%
% \bbl@provide@dirs{#2}%
% \bbl@csarg\ifnum{\wdir@#2}>\z@
% \let\bbl@beforeforeign\leavevmode
% \EnableBabelHook{babel-bidi}%
% \fi
% \bbl@foreach{#2}{%
% \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
% \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
% \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}
% \def\bbl@fsstore[#1]{(rm)\rmdefault\bbl@save@rmdefault}
% \def\bbl@fsstore[#1]{(af)\afdefault\bbl@save@afdefault}
% \def\bbl@fsstore[#1]{(tt)\ttdefault\bbl@save@ttdefault}
% \def\bbl@fsstore[#1]{#2#3#4}%
% \bbl@csarg\edef\famrst@#4{#3}%
% \expandafter\addto\csname extras\#1\endcsname{%
% \let#4#3%
% \ifx#3\@family
% \edef#3{\csname babl@#2default@#1\endcsname}%
% \fontfamily{#3}\selectfont
% \else
% \edef#3{\csname babl@#2default@#1\endcsname}%
% \fi
% \expandafter\addto\csname noextras\#1\endcsname{%
% \fi
% \let#3#4}
% \let\bbl@langfeatures@empty
12 Hooks for XeTeX and LuaTeX

12.1 XeTeX

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

155
Now, the code.
\begin{xetex}
\def\BabelStringsDefault{\unicode}
\let\xebl@stop\relax
\AddBabelHook{xetex}{encodedcommands}{%}
\def\bbl@tempa{#1}\
\ifx\bbl@tempa\@empty
\XeTeXinputencoding{bytes}\
\else
\XeTeXinputencoding{#1}\
\fi
\def\xebl@stop{\XeTeXinputencoding{utf8}}%
\AddBabelHook{xetex}{stopcommands}{%}
\xebl@stop
\let\xebl@stop\relax
\def\bbl@intraspace#1 #2 #3\@@{%}
\bbl@csarg{xeisp@languagename}%=\XeTeXlinebreakskip #1em plus #2em minus #3em\relax
\def\bbl@intrapenalty#1\@@{%}
\bbl@csarg{xeipn@languagename}%=\XeTeXlinebreakpenalty #1\relax
\def\bbl@provide@intraspace{%}
\bbl@ixin{/s}{/\bbl@cl{lnbrk}}\
\ifin@\else\bbl@ixin{/c}{/\bbl@cl{lnbrk}}\fi
\ifin@
\bbl@ifunset{bbl@intsp@languagename}{}\
\ifx\csname bbl@intsp@languagename\endcsname\@empty\else
\ifx\bbl@KVP@intraspace\@nil
\bbl@exp{\bbl@intraspace\bbl@KVP@intraspace\@@\@@}\
\fi
\ifx\bbl@KVP@intrapenalty\@nil
\bbl@intrapenalty0\@@\@@\@@
\fi
\fi
\\expandafter\ifx\csname bbl@KVP@intrapenalty\endcsname\@nil\else % We may override the ini
\expandafter\ifx\csname bbl@KVP@intrapenalty\endcsname\@nil\else % We may override the ini
% TODO. Execute only once (but redundant):
\\bbl@add<extras@languagename>%
\XeTeXlinebreaklocale "\bbl@cl{tbcp}"
\bbl@add<noextras@languagename>%
\XeTeXlinebreaklocale "en"
\fi
\expandafter\ifx\csname bbl@ispacesize\endcsname\@undefined
\gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
\expandafter\ifx\csname bbl@patchfont\endcsname\@notprerr
\expandafter\@secondoftwo % to execute right now
\fi
\AtBeginDocument\motprerr
\expandafter\ifx\csname bbl@patchfont\endcsname\@notprerr
\AtBeginDocument{\@secondoftwo % to execute right now
\fi
\fi\fi}
\end{xetex}
12.2 Layout

In progress.

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

Consider \comment{txtbabel} as a shorthand for \comment{tex–xet}, which is the bidi model in both pdftex and xetex.
12.3 LuaTeX

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

The names \texttt{\l@<language>} are defined and take some value from the beginning because all \texttt{ldf} files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first selected (which usually means when the \texttt{ldf} finishes). If a language has been loaded, \texttt{\bbl@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 \texttt{language.dat} have the same name then just ignore the latter. If there are new synonymous, the are added, but note if the language patterns have not been preloaded they won't at run time.

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

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

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

We need catcode tables, but no format (targeted by babel) provide a command to allocate them (although there are packages like 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 file is read at three places: (1) when plain.def, babel.sty starts, to read the list of available languages from Language.dat (for the base option); (2) at hyphen.cfg, to modify some macros; (3) in the middle of plain.def and babel.sty, by babel.def, with the commands and other definitions for luatex (eg. \babelpatterns).
\def\bbl@patterns@lua#1{\%
\language=\ifx\csname l@#1:\f@encoding\endcsname\relax
\csname l@#1\endcsname
\edef\bbl@tempa{#1}\
\else
\csname l@#1:\f@encoding\endcsname
\edef\bbl@tempa{#1:\f@encoding}\
\fi\relax
\ifdefined{luatexhyphen@loaded@\the\language}{\% Temp
@ifundefined{bbl@hyphendata@\the\language}{\{\{#1\}#4\}#3}\%
\iffalse\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}{}\%
\{\{#1\}#2\}\%
\fi\%
\bbl@languages
@ifundefined{bbl@hyphendata@\the\language}{\{\{#1\}#2\}#4\}#3\%
\{\{#1\}#4\}
\}{\{#3}#4\}#2\%
\{\{#1\}#3\}#1\%
\{\{#1\}#4\}
\}{\{#3}#4\}#2\%
\{\{#1\}#5\}#6\%
\{\{#1\}#4\}#3\%
\{\{#2}#4\}
\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##4{\%
\def\process@line####1####2####3####4{}\%
\AddBabelHook{luatex}{loadpatterns}{}\%
\{\{#1\}#2\}#3\%
\{\{#1\}#4\}#3\%
\{\{#1\}#5\}#4\%
\{\{#1\}#6\}#4\%
\}{\{#3}#4\}#2\%
\{\{#1\}#3\}#1\%
\{\{#1\}#4\}#2\%
\{\{#1\}#5\}#1\%
\{\{#1\}#6\}#1\%
\{\{#1\}#7\}#0\%
\}{\{#3}#4\}#3\%
\{\{#2}#4\}
\endinput\fi
% Here stops reading code for hyphen.cfg
% The following is read the 2nd time it's loaded
\begingroup % TODO - to a lua file
\catcode`%=12
\catcode`\%=12
\catcode`\%=12
\catcode`\%=12
\directlua{
Babel = Babel or {}\%
function Babel.bytes(line)
  return line:gsub("(.)",function(chr) return unicode.utf8.char(string.byte(chr)) end)
end\%
function Babel.begin_process_input()
  if luatexbase and luatexbase.add_to_callback then
    luatexbase.add_to_callback('process_input_buffer',Babel.bytes)
  else
    Babel.callback = callback.find('process_input_buffer')\%
    callback.register('process_input_buffer',Babel.bytes)
  end
\endinput\fi
}
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("
                \[
                    \texttt{\textbf{bbl@info\endcsname{New pattern: }}\hspace{1em} \texttt{}}
                \]
            pats = pats .. ' ' .. p
        else
            tex.sprint("
                \[
                    \texttt{\textbf{bbl@info\endcsname{Renew pattern: }}\hspace{1em} \texttt{}}
                \]
            end
        end
    lang.patterns(lg, pats)
end

function Babel.hlist_has_bidi(head)
    local has_bidi = false
    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(Babel.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

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("
                \[
                    \texttt{\textbf{bbl@info\endcsname{New pattern: }}\hspace{1em} \texttt{}}
                \]
            pats = pats .. ' ' .. p
        else
            tex.sprint("
                \[
                    \texttt{\textbf{bbl@info\endcsname{Renew pattern: }}\hspace{1em} \texttt{}}
                \]
            end
        end
    lang.patterns(lg, pats)
end

function Babel.hlist_has_bidi(head)
    local has_bidi = false
    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(Babel.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

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("
                \[
                    \texttt{\textbf{bbl@info\endcsname{New pattern: }}\hspace{1em} \texttt{}}
                \]
            pats = pats .. ' ' .. p
        else
            tex.sprint("
                \[
                    \texttt{\textbf{bbl@info\endcsname{Renew pattern: }}\hspace{1em} \texttt{}}
                \]
            end
        end
    lang.patterns(lg, pats)
end

function Babel.hlist_has_bidi(head)
    local has_bidi = false
    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(Babel.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

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("
                \[
                    \texttt{\textbf{bbl@info\endcsname{New pattern: }}\hspace{1em} \texttt{}}
                \]
            pats = pats .. ' ' .. p
        else
            tex.sprint("
                \[
                    \texttt{\textbf{bbl@info\endcsname{Renew pattern: }}\hspace{1em} \texttt{}}
                \]
            end
        end
    lang.patterns(lg, pats)
end

function Babel.hlist_has_bidi(head)
    local has_bidi = false
    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(Babel.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

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

12.4 Southeast Asian scripts
First, some general code for line breaking, used by \babelposthyphenation.
Replace regular (i.e., implicit) discretionaries by spaceskips, based on the previous glyph (which I
think makes sense, because the hyphen and the previous char go always together). Other
discretionaries are not touched. See Unicode UAX 14.
\% TODO - to a lua file
\begin{verbatim}
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)
tex.print(([[\noexpand\csname bbl@luahyphenate\endcsname]]))
table.insert(Babel.linebreaking.before, func)
end
function Babel.linebreaking.add_after(func)
tex.print(([[\noexpand\csname bbl@luahyphenate\endcsname]]))
table.insert(Babel.linebreaking.after, func)
end
\end{verbatim}
\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
}\end{verbatim}
\begin{verbatim}
\endgroup
\end{verbatim}

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..' ', '(.-)%d.(.-)%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}

12.5 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'
    }\"bbl@luahyphenate}
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\m@nohyphenation\space and
      last_lang == \the\m@nohyphenation then
      local intrapenalty = props.intrapenalty
      if intrapenalty ~= 0 then
        local n = node.new(14, 0) % penalty
        n.penalty = intrapenalty
        node.insert_before(head, item, n)
      end
      local intraspace = props.intraspace
      local n = node.new(12, 13) % (glue, spaceskip)
      node.setglue(n, intraspace.b * quad,
                    intraspace.p * quad,
                    intraspace.m * quad)
      node.insert_before(head, item, n)
    end
    if font.getfont(item.font) then
      quad = font.getfont(item.font).size
    end
    last_class = class
    last_lang = lang
  else % if penalty, glue or anything else
    last_class = nil
  end
  lang.hyphenate(head)
end

\bbl@luahyphenate{}
def\bbl@luahyphenate{}
\let\bbl@luahyphenate\relax
\directlua{
luatexbase.add_to_callback('hyphenate',
  function (head, tail)
    %
    \bbl@luahyphenate}
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}\\@@}\if\bbl@KVP@intrapenalty\@nil\bbl@intrapenalty0\@@\fi\else % sea\bbl@seaintraspace\bbl@exp{\\bbl@intraspace\\bbl@cl{intsp}\\@@}\directlua{\bbl@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}\bblar@elongated{\bbl@bblar@chars{}}\if\bbl@KVP@intrapenalty\@nil\bbl@intrapenalty0\@@\fi\else\expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@\fi}}

12.6 Arabic justification
\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{%
\begingroup
\catcode`_=11 \catcode`:11
\gdef\bblar@nofswarn{\gdef\msg_warning:nnx##1##2##3{}}
\endgroup
\gdef\bbl@arabicjust{%
  \let\bbl@arabicjust\relax
  \newattribute\bblar@kashida
  \directlua{ Babel.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{^^^^200d\char"##1#2}}%
      \setbox\z@\hbox{^^^^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
      }}%
    % 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] == nil then
            Babel.arabic.elong_map[[the\localeid][fontid] = {}
            tex.print([[\string\csname bbl\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{%}
    \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
    \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
    % \addfontfeature{RawFeature=+jalt}% todo: catch medial kaf
\endgroup
\begingroup
\catcode`_=11 \catcode`:11
\gdef\bblar@nofswarn\gdef\msg_warning:nnx##1##2##3{}}
\endgroup
\gdef\bbl@arabicjust%
  \let\bbl@arabicjust\relax
  \newattribute\bblar@kashida
  \directlua{ Babel.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{^^^^200d\char"##1#2}}%
      \setbox\z@\hbox{^^^^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
      }}%
    % 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] == nil then
            Babel.arabic.elong_map[[the\localeid][fontid] = {}
            tex.print([[\string\csname bbl\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{%}
    \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
    \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
    % \addfontfeature{RawFeature=+jalt}% todo: catch medial kaf
\endgroup
\begingroup
\catcode`_=11 \catcode`:11
\gdef\bblar@nofswarn\gdef\msg_warning:nnx##1##2##3{}}
\endgroup
\gdef\bbl@arabicjust%
  \let\bbl@arabicjust\relax
  \newattribute\bblar@kashida
  \directlua{ Babel.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{^^^^200d\char"##1#2}}%
      \setbox\z@\hbox{^^^^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
      }}%
    % 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] == nil then
            Babel.arabic.elong_map[[the\localeid][fontid] = {}
            tex.print([[\string\csname bbl\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{%}
    \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
    \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
    % \addfontfeature{RawFeature=+jalt}% todo: catch medial kaf
\endgroup
\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}

\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
}

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 last_line
  local GLYPH = node.id'glyph'
  local KASHIDA = Babel.attr_kashida
  local LOCALE = Babel.attr_locale

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

% Exclude last line. todo. But-- it discards one-word lines, too!
% ? Look for glue = 12:15
if (line.glue_sign == 1 and line.glue_order == 0) then
  elongs = {} % Stores elongated candidates of each line
  k_list = {} % And all letters with kashida
  pos_inline = 0 % Not yet used

  for n in node.traverse_id(GLYPH, line.head) do
    pos_inline = pos_inline + 1 % To find where it is. Not used.
    % Elongated glyphs
    if elong_map then
      local locale = node.get_attribute(n, LOCALE)
      if elong_map[locale] and elong_map[locale][n.font] and
         elong_map[locale][n.font][n.char] then
        table.insert(elongs, {node = n, locale = locale} )
        node.set_attribute(n.prev, KASHIDA, 0)
      end
    end
  end

  % 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 = elongss[x].node
  local oldchar = curr.char
  curr.char = elong_map[curr.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
wt_pos = 1
while width < goal do
  subst_done = true
  k_item = k_list[k_curr].node
  if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
    d = node.copy(k_item)
    d.char = 0x0640
    line.head, new = node.insert_after(line.head, k_item, d)
    width_new = node.dimensions(line.head)
    if width > goal or width == width_new then
      node.remove(line.head, new) % Better compute before
      break
    end
    width = width_new
  end
  if k_curr == 1 then
    k_curr = #k_list
    wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
  else
    k_curr = k_curr - 1
  end
end::next_line::

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

% 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}},
  }
} % (Font selection)
['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
['Cyr'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F}},
['Dev'] = {{0x0900, 0x097F}, {0x0A80, 0x0A8F}},
['Eth'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x1C80, 0x1C8F},
{0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
['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'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
['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, 0x9FF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
{0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
['Laoo'] = {{0x0E80, 0x0EFF}},
['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
{0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
{0x2CEB0, 0x2EEFF}, {0x2F800, 0x2FA1F}},
['Mahj'] = {{0x11150, 0x1117F}},
['Mlym'] = {{0x0D00, 0x0D7F}},
['Mymr'] = {{0x1000, 0x109F}, {0x0AA60, 0x0A7F}, {0xA900, 0xA9FF}},
['Orya'] = {{0x0B00, 0x0B7F}},
['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
['Syr'] = {{0x0E00, 0x0E7F}},
['Taml'] = {{0x0B80, 0x0BFF}},
['Telu'] = {{0x0C00, 0x0C7F}},
['Tfng'] = {{0x0D30, 0x0D7F}},
['Thai'] = {{0x0E00, 0x0E7F}},
['Tibt'] = {{0x0F00, 0x0FF}},
['Vaii'] = {{0xA500, 0xA63F}},
['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}}

Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl

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 i, scr in pairs(maps) do
            if toloc[i] then
              GLYPH = scr
              break
            end
          end
        end
      end
    end
  end
end

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 i, scr in pairs(maps) do
          if toloc[i] then
            GLYPH = scr
            break
          end
        end
      end
    end
  end
end
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

% Now, take action, but treat composite chars in a different
% fashion, because they 'inherit' the previous locale. Not yet
% optimized.
if not toloc and
  (item.char >= 0x0300 and item.char <= 0x036F) or
  (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
  (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
  toloc = toloc_save
end
if toloc and toloc > -1 then
  if Babel.locale_props[toloc].lg then
    item.lang = Babel.locale_props[toloc].lg
    node.set_attribute(item, LOCALE, toloc)
  end
  if Babel.locale_props[toloc] ['/..item.font] then
    item.font = Babel.locale_props[toloc] ['/..item.font]
  end
  toloc_save = toloc
end
elseif not inmath and item.id == 7 then
  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
}
}

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

\newcommand\babelcharproperty[1]{%
  \count@=#1\relax
  \ifvmode
    \bbl@chprop
  \else
    \bbl@error{\string\babelcharproperty\space can be used only in\% vertical mode (preamble or between paragraphs)}%
    \end{tabbing}
    \{\See the manual for futher info\%}
  \fi}
\newcommand\bbl@chprop[3][\the\count@]{%\tempcnta=#1\relax
  \bbl@ifunset{bbl@chprop@#2}{%
    \bbl@error{No property named '#2'. Allowed values are\% direction (bc), mirror (bmg), and linebreak (lb)}%
    \{\See the manual for futher info\%}
  }%
  \loop
    \bbl@cs{chprop@#2}{#3}
    \ifnum\count@<\@tempcnta
      \advance\count@\@ne
    \repeat}
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.

```
\def\bbl@chprop@direction#1{%
  \directlua{
    Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
    Babel.characters[\the\count@][\textquoteleft d\textquoteright] = '#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@][\textquoteleft m\textquoteright] = \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@][\textquoteleft c\textquoteright] = '#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
  }
}\let\bbl@chprop@locale\bbl@chprop@linebreak
\def\bbl@chprop@locale{\the\l@nohyphenation}
\begingroup
  \catcode\~=12
  \catcode\%=12
  \catcode\&=14
  \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
    \else
      \bbl@activateposthyphen
    \fi
    \begingroup
      \def\babeltempa{\bbl@add@list\babeltempb}
      \let\babeltempb\@empty
      \def\bbl@tempa{#5}
      \bbl@replace\bbl@tempa{,}{ ,}
      \expandafter\bbl@foreach\expandafter{\bbl@tempa}{
        \bbl@ifsamestring{##1}{remove}{\bbl@add@list\babeltempb{nil}}
        \directlua{
          local rep = [=[\textquoteleft##1\textquoteright]=]}
    \endgroup
  }
\endgroup
```

Now the \TeX\ high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the \texttt{(n)} syntax. For example, \texttt{pre={1}{1}-} becomes \texttt{function(m) return m[1]..m[1]..'-' end}, where \texttt{m} are the matches returned after applying the pattern. With a mapped capture the functions are similar to \texttt{function(m) return Babel.capt_map(m[1],1) end}, where the last argument identifies the mapping to be applied to \texttt{m[1]}. The way it is carried out is somewhat tricky, but the effect in not dissimilar to \texttt{lua \texttexttt{load}} – save the code as string in a \TeX\ macro, and expand this macro at the appropriate place. As \texttt{\directlua} does not take into account the current catcode of \texttt{&}, we just avoid this character in macro names (which explains the internal group, too).
rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
rep = rep:gsub('^%s*(insert)%s*,', 'insert = true, ')
rep = rep:gsub('(string)%s*=%s*(["%s,%"]*)', Babel.capture_func)
if #1 == 0 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 .. }')
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 .. [\[}}]])
}}&%
\let\bbl@kv@attribute\relax
\let\bbl@kv@label\relax
\bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}&%
\ifx\bbl@kv@attribute\relax\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
\if\bbl@kv@attribute\relax\relaxelse &% 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 })
}&%
As a first step, add a handler for bidi and digits (and potentially other processes) just before luatexofload is applied, which is loaded by default by \LaTeX. Just in case, consider the possibility it has not been loaded.

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

\begin{verbatim}
12.9 Bidi

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

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

\end{verbatim}
\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')
fi}
% TODO - to locale_props, not as separate attribute
\newattribute\bbl@attr@dir
\directlua{ Babel.attr_dir = luatexbase.registernumber'\bbl@attr@dir' }
% TODO. I don't like it, hackish:
\bbl@exp{\output{\bodydir\pagedir\the\output}}
\AtEndOfPackage{\EnableBabelHook{babel-bidi}}
\fi
\chardef\bbl@thetextdir\z@
\chardef\bbl@thepardir\z@
\def\bbl@getluadir#1{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
\ifcase#3\relax
\ifcase\bbl@getluadir{#1}\relax
#2 TLT\relax
\else
\ifcase\bbl@getluadir{#1}\relax
#2 TRT\relax
\fi
\else
\fi}
\def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
\ifcase#3\relax
\ifcase\bbl@getluadir{#1}\relax
#2 TLT\relax
\else
\ifcase\bbl@getluadir{#1}\relax
#2 TRT\relax
\fi
\else
\fi
\fi
\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*3+#1}\
\setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
\def\bbl@pardir#1{\bbl@setluadir{par}\pardir{#1}\
\chardef\bbl@thepardir#1\relax}
\def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
 Unlike \textit{xetex}, \textit{luatex} requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with \texttt{bidi=basic}, without having to patch almost any macro where text direction is relevant.

\texttt{@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 \texttt{\bodydir}), and when \texttt{\parbox} and \texttt{\hangindent} are involved. Fortunately, latest releases of \texttt{luatex} simplify a lot the solution with \texttt{\shapemode}.

With the issue \#15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, tabular seems to work (at least in simple cases) with \texttt{array}, \texttt{tabularx}, \texttt{hhline}, \texttt{colortbl}, \texttt{longtable}, \texttt{booktabs}, etc. However, \texttt{dcolumn} still fails.

\begin{verbatim}
5872        d.dir = '+TRT'
5873        node.insert_before(head, node.has_glyph(head), d)
5874        for item in node.traverse(head) do
5875            node.set_attribute(item,  # Babel.attr_dir, token.get_macro('bbl@thedir'))
5876        end
5877    end
5878    return head
5879 end
5880 return head
5881 end
5882 luatexbase.add_to_callback(  # "Babel.math_box_dir", 0)
5883 })%
5884 \fi

12.10 Layout

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

\texttt{@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 \texttt{\bodydir}), and when \texttt{\parbox} and \texttt{\hangindent} are involved. Fortunately, latest releases of \texttt{luatex} simplify a lot the solution with \texttt{\shapemode}.

With the issue \#15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, \texttt{tabular} seems to work (at least in simple cases) with \texttt{array}, \texttt{tabularx}, \texttt{hhline}, \texttt{colortbl}, \texttt{longtable}, \texttt{booktabs}, etc. However, \texttt{dcolumn} still fails.

\begin{verbatim}
5886 \bbl@trace{Redefinitions for bidi layout}
5887 \def\BabelNoAMSMath{\let\bbl@noamsmath\relax}
5888 \ifnum\bbl@bidimode>\z@
5889 \let\bbl@eqnodir\relax
5890 % \ifx\matheqdirmode\@undefined\else
5891 % \matheqdirmode\@ne
5892 % \fi
5893 \AtBeginDocument{%
5894 \ifx\maketag@@@\@undefined \else\% Normal equation, eqnarray
5895 \AddToHook{env/eqnarray/begin}{%  #178
5896 \ifnum\bbl@thetextdir>\z@%
5897 \def\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
5898 \chardef\bbl@thetextdir\z@%
5899 \bbl@add\normalfont{\bbl@eqnodir}%
6000 \pardir TLT % dir for \texttt{\eqno} is \texttt{\pardir}!
6001 \fi}%
6002 \AddToHook{env/equation/begin}{%
6003 \ifnum\bbl@thetextdir>\z@%  #178
6004 \def\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6005 \chardef\bbl@thetextdir\z@%
6006 \bbl@add\normalfont{\bbl@eqnodir}%
6007 \pardir TLT \% dir for \texttt{\eqno} is \texttt{\pardir}!
6008 \fi}%
6009 \bbl@xin@{,leqno},{,\@classoptionslist},%
6010 \ifin@
6011 \def\eqnum{0}
6012 \setbox\z@\hbox{\normalfont\normalcolor{\theequation}}%
6013 \hbox to0.01pt{\hss\box\z@\hss}%%%
6014 \else % amstex
6015 \ifx\bbl@noamsmath\undefined
6016 \ifpackagewith{amsmath}{leqno}%
6017 \begin{eqnarray}{C}
6018 \def\bbl@ams@lap\box{\hbox}{ leqno
6019 \def\bbl@ams@lap\box{\llap}{ default
6020 \ExplSyntaxOn
6021 \end{verbatim}

178
Implicitly reverses sectioning labels in bidi=basic-r, because the full stop is not in contact with L numbers any more. I think there must be a better way. Assumes bidi=basic, but there are some additional readjustments for bidi=default.
Some \LaTeX{} macros use internally the math mode for text formatting. They have very little in common and are grouped here, as a single option.

\begin{verbatim}
\let\bbl\asciiroman\@roman
\let\bbl\OL\@roman
\def\@roman#1{\babelsublr{\ensureascii{\bbl\asciiroman#1}}}
\let\bbl\asciiRoman\@Roman
\let\bbl\OL\@Roman
\def\@Roman#1{\babelsublr{\ensureascii{\bbl\asciiRoman#1}}}
\let\bbl\OL\labelenumii\labelenumii
\def\labelenumii{\)\theenumii(}
\let\bbl\OL\underline\underline
\bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}
\let\bbl\OL\LaTeX2e\LaTeX2e
\DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
\if b\expandafter\@car\f@series\@nil\boldmath\fi
\babelsublr{\LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
\end{verbatim}

\section{12.11 Lua: transforms}

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

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

\begin{verbatim}
\let\bbl\OL\@footnote\footnote
\BabelFootnote\footnote\languagename{}{}
\BabelFootnote\localfootnote\languagename{}{}
\BabelFootnote\mainfootnote{}{}
\end{verbatim}
n.char  = s
if not head then
  head = n
else
  last.next = n
end
last = n
end
return head
end
Babel.fetch_subtext = {}
Babel.ignore_pre_char = function(node)
  return (node.lang == Babel.nohyphenation)
end
-- Merging both functions doesn't seem feasible, because there are too
-- many differences.
Babel.fetch_subtext[0] = function(head)
  local word_string = ''
  local word_nodes = {}
  local lang
  local item = head
  local inmath = false
  while item do
    if item.id == 11 then
      inmath = (item.subtype == 0)
    end
    if inmath then
      -- pass
    elseif item.id == 29 then
      local locale = node.get_attribute(item, Babel.attr_locale)
      if lang == locale or lang == nil then
        lang = lang or locale
      end
      if Babel.ignore_pre_char(item) then
        word_string = word_string .. Babel.us_char
      else
        word_string = word_string .. unicode.utf8.char(item.char)
      end
      word_nodes[#word_nodes+1] = item
    elseif item.id == 12 and item.subtype == 13 then
      word_string = word_string .. ' ' 
      word_nodes[#word_nodes+1] = item
    else
      break
    end
    item = item.next
  end
end
else 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
  end
  if Babel.ignore_pre_char(item) then
    word_string = word_string .. Babel.us_char
  else
    word_string = word_string .. unicode.utf8.char(item.char)
  end
  word_nodes[#word_nodes+1] = item
  else
    break
  end
elseif item.id == 12 and item.subtype == 13 then
  word_string = word_string .. ' '
  word_nodes[#word_nodes+1] = item
else
  -- 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
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

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
    elseif word_string == '' then
      -- pass
    elseif (item.id == 12 and item.subtype == 13) then
      break
    else
      word_string = word_string .. Babel.us_char
      word_nodes[#word_nodes+1] = item -- Will be ignored
    end
    item = item.next
  end
  word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
  return word_string, word_nodes, item, lang
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
      local last_match = 0
      local step = 0
      -- For every match.
      while true do
        if Babel.debug then
          print('=====
        end
        local new -- used when inserting and removing nodes
        local matches = { u.match(w, p, last_match) }
        if #matches < 2 then break end
        -- Get and remove empty captures (with ()'s, which return a
        -- number with the position), and keep actual captures
        -- (from (...)), if any, in matches.
local first = table.remove(matches, 1)
local last = table.remove(matches, #matches)
-- Non re-fetched substrings may contain \31, which separates
-- subsubstrings.
if string.find(w:sub(first, last-1), Babel.us_char) then break end

local save_last = last -- with A()BC()D, points to D
-- Fix offsets, from bytes to unicode. Explained above.
first = u.len(w:sub(1, first-1)) + 1
last = u.len(w:sub(1, last-1)) -- now last points to C

-- This loop stores in a small table the nodes
-- corresponding to the pattern. Used by 'data' to provide a
-- predictable behavior with 'insert' (w_nodes is modified on
-- the fly), and also access to 'remove'd nodes.
local sc = first-1 -- Used below, too
local data_nodes = {}  

local enabled = true
for q = 1, last-first+1 do
    data_nodes[q] = w_nodes[sc+q]
    if enabled
        and attr > -1
        and not node.has_attribute(data_nodes[q], attr)
    then
        enabled = false
    end
end

-- This loop traverses the matched substring and takes the
-- corresponding action stored in the replacement list.
-- sc = the position in substr nodes / string
-- rc = the replacement table index
local rc = 0

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

    local crep = r[rc]
    local item = w_nodes[sc]
    local item_base = item
    local placeholder = Babel.us_char
    local d
if crep and crep.data then
    item_base = data_nodes[crep.data]
end

if crep then
    step = crep.step or 0
end

if (not enabled) or (crep and next(crep) == nil) then -- = {}
    last_match = save_last -- Optimization
    goto next
elseif crep == nil or crep.remove then
    node.remove(head, item)
    table.remove(w_nodes, sc)
    w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
    sc = sc - 1 -- Nothing has been inserted.
    last_match = utf8.offset(w, sc+1+step)
    goto next
elseif crep and crep.kashida then -- Experimental
    node.set_attribute(item, Babel.attr_kashida, crep.kashida)
    last_match = utf8.offset(w, sc+1+step)
    goto next
elseif crep and crep.string then
    local str = crep.string(matches)
    if str == '' then -- Gather with nil
        node.remove(head, item)
        table.remove(w_nodes, sc)
        w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
        sc = sc - 1 -- Nothing has been inserted.
    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
if mode == 1 and crep and (crep.pre or crep.no or crep.post) then
    d = node.new(7, 0) -- (disc, discretionary)
    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
        d.placeholder = '|'  
        head, new = node.insert_before(head, item, d)
    else
        d.penalty = crep.penalty or tex.exhyphenpenalty
        head, new = node.insert_before(head, item, d)
    end
else if mode == 0 and crep and (crep.pre or crep.no or crep.post) then  
    -- ERROR
else if 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)
else if 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
    if mode == 0 then
        placeholder = ' ' 
        end
    head, new = node.insert_before(head, item, d)
else if crep and crep.spacefactor then
    d = node.new(12, 13) -- (glue, spaceskip)
    local base_font = font.getfont(item_base.font)
    if mode == 0 then
        placeholder = ' ' 
        end
    head, new = node.insert_before(head, item, d)
else if mode == 0 and crep and crep.space then
    -- ERROR
end -- ie replacement cases
-- Shared by disc, space and penalty.
if sc == 1 then
    word_head = head
end
if crep.insert then
    w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
    table.insert(w_nodes, sc, new)
    last = last + 1
else
    w_nodes[sc] = d
    node.remove(head, item)
w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
end
last_match = utf8.offset(w, sc+1*step)
::next::
end -- for each replacement
if Babel.debug then
    print('.....', '/')
    Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
end
end -- for match
::next::
word_head = nw
end -- for substring
end -- for patterns
::next::
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('{(%x%x%x%x+)}', function (n)
        return u.char(tonumber(n, 16))
    end)
    ret = ret:gsub("%\[\%\[\]%\]%.%.", '')
    ret = ret:gsub("%.%.%\[\%\[\]%\]", '')
    return key .. [[=function(m) return \]] .. ret .. [[ end]]
end

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

-- 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
    for s in string.utfcharacters(to) do
        table.insert(froms, s)
    end
    return froms
end

189
local cnt = 1
local mlen = table.getn(Babel.capture_maps)
for s in string.utfcharacters(to) do
  Babel.capture_maps[mlen][froms[cnt]] = s
  cnt = cnt + 1
end
return ']].Babel.capt_map(m[" .. capno .. "]," .. (mlen) .. ") .. "][:,:,Babel.capt_map(m[" .. capno .. "]," .. (mlen) .. ") .. "]

-- 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

12.12 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=0x29},
[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.

```
6623 (+basic-r)
6624 Babel = Babel or {}
6625 Babel.bidi_enabled = true
6627 require('babel-data-bidi.lua')
6629 local characters = Babel.characters
6631 local ranges = Babel.ranges
6632 local DIR = node.id("dir")
6633 local function dir_mark(head, from, to, outer)
6634   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6635   local d = node.new(DIR)
6636   d.dir = '+' .. dir
6637   node.insert_before(head, from, d)
6638   d = node.new(DIR)
6639   d.dir = '-' .. dir
6640   node.insert_after(head, to, d)
6641 end
6642 function Babel.bidi(head, ispar)
6643   local first_n, last_n -- first and last char with nums
6644   local last_es -- an auxiliary 'last' used with nums
6645   local first_d, last_d -- first and last char in L/R block
6646   local dir, dir_real
6647   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6648   local strong_lr = (strong == 'l') and 'l' or 'r'
6649   local outer = strong
6650   local new_dir = false
6651   local first_dir = false
6652   local inmath = false
6653   local last_lr
6654   local type_n = ''
6655   for item in node.traverse(head) do
6656     -- three cases: glyph, dir, otherwise
6657     if item.id == node.id('glyph') then
6658       local itemchar
6659       if item.id == 7 and item.subtype == 2 then
6660         itemchar = item.replace.char
6661       else
6662         itemchar = item.char
6663       end
6664       local chardata = characters[itemchar]
```
dir = chardata and chardata.d or nil
if not dir then
    for nn, et in ipairs(ranges) do
        if itemchar < et[1] then
            break
        elseif itemchar <= et[2] then
            dir = et[3]
            break
        end
    end
    dir = dir or 'l'
end
if inmath then dir = ('TRT' == tex.mathdir) and 'r' or 'l' end

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 % 3
        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.

if strong == '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
elif dir == 'es' and last_n then -- W3+W6
    last_es = item
elif dir == 'cs' then -- it's right - do nothing
    elseif dir 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_d, last_d, outer)
            first_d, last_d = nil, nil
        elseif strong_lr == 'l' and type_n ~= '' then
            last_d = last_n
        end
    elseif dir then -- ''
        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, whatsit's, 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>, respictly, but with other combinations depends on outer. From all these, we select only those resolving <on> → <r>. At the beginning (when last_lr is nil) of an R text, they are mirrored directly.

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

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

end

⟨/basic⟩

And here the Lua code for bidi=basic:

Babel = Babel or {}

-- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
Babel.fontmap = Babel.fontmap or {}
Babel.fontmap[0] = {} -- l
Babel.fontmap[1] = {} -- r

Babel.bidi_enabled = true
Babel.mirroring_enabled = true
require('babel-data-bidi.lua')
local characters = Babel.characters
local ranges = Babel.ranges
local DIR = node.id('dir')
local GLYPH = node.id('glyph')
local function insert_implicit(head, state, outer)
  local new_state = state
  if state.sim and state.eim and state.sim ~= state.eim then
    dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
    local d = node.new(DIR)
    d.dir = '+' .. dir
    node.insert_before(head, state.sim, d)
    d = node.new(DIR)
    d.dir = '-' .. dir
    node.insert_after(head, state.eim, d)
  end
  new_state.sim, new_state.eim = nil, nil
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

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

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

    local glue_d = nil
    local glue_i = nil

    local has_en = false
    local first_et = nil

    local ATDIR = Babel.attr_dir
    local save_outer
    local temp = node.get_attribute(head, ATDIR)
    if temp then
        temp = temp % 3
        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 item.id == GLYPH
    or (item.id == 7 and item.subtype == 2) then
    local d_font = nil
    local item_r
    if item.id == 7 and item.subtype == 2 then
      item_r = item.replace -- automatic discs have just 1 glyph
    else
      item_r = item
    end
    local chardata = characters[item_r.char]
    d = chardata and chardata.d or nil
    if not d or d == 'nsm' then
      for nn, et in ipairs(ranges) do
        if item_r.char < et[1] then
          break
        elseif item_r.char <= et[2] then
          if not d then d = et[3]
          elseif d == 'nsm' then d_font = et[3]
          end
        end
        break
      end
      d = d or 'l'
    end
    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
  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 % 3
    end
    if attr_d == 1 then
      outer_first = 'l'
    elseif attr_d == 2 then
      outer_first = 'r'
    else
      outer_first = 'l'
    end
  end
last = 'l'
end
outer = last
has_en = false
first_et = nil
new_d = false
end
if glue_d then
  if (d == 'l' and 'l' or 'r') ~= glue_d then
    table.insert(nodes, {glue_i, 'on', nil})
  end
  glue_d = nil
  glue_i = nil
end
elseif item.id == DIR then
  d = nil
  if head -= item then new_d = true end
elseif item.id == node.id'glue' and item.subtype == 13 then
  glue_d = d
  glue_i = item
  d = nil
elseif item.id == node.id'math' then
  inmath = (item.subtype == 0)
else
  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 nodes[e][1].id == GLYPH then nodes[e][2] = temp end
end
first_et = nil
has_en = false
end
-- Force mathdir in math if ON (currently works as expected only
-- with 'l')
if inmath and d == 'on' then
    d = ('TRT' == tex.mathdir) and 'r' or 'l'
end
if d then
    if d == 'al' then
        d = 'r'
        last = 'al'
    elseif d == 'l' or d == 'r' then
        last = d
    end
    prev_d = d
    table.insert(nodes, {item, d, outer_first})
end
outer_first = nil
end
-- TODO -- repeated here in case EN/ET is the last node. Find a
-- better way of doing things:
if first_et then -- dir may be nil here!
    if has_en then
        if last == 'l' then
            temp = 'l'  -- W7
        else
            temp = 'en'  -- W5
        end
    else
        temp = 'on'  -- W6
    end
    for e = first_et, #nodes do
        if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
    end
end
-- dummy node, to close things
table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
-------------- NEUTRAL ---------------
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
if d == 'on' then
first_on = first_on or q
elseif first_on then
if last == d then
  temp = d
else
  temp = outer
end
for r = first_on, q - 1 do
  nodes[r][2] = temp
  item = nodes[r][1] -- MIRRORING
  if Babel.mirroring_enabled and item.id == GLYPH
     and temp == 'r' and characters[item.char] then
    local font_mode = ''
    if 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
  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
if d == 'en' then d = 'an' end
local isdir = (d == 'r' or d == 'l')
if outer == 'l' and d == 'an' then
  state.san = state.san or item
  state.ean = item
elseif state.san then
  state.san = state.san or item
  state.ean = item
  state.has_r = true
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
if d == 'en' then d = 'an' end
local isdir = (d == 'r' or d == 'l')
if outer == 'l' and d == 'an' then
  state.san = state.san or item
  state.ean = item
elseif state.san then
  state.san = state.san or item
  state.ean = item
  state.has_r = true
end
end
7141    head, state = insert_numeric(head, state)
7142    end
7143
7144    if outer == 'l' then
7145        if d == 'an' or d == 'r' then  -- im -> implicit
7146            if d == 'r' then state.has_r = true end
7147            state.sim = state.sim or item
7148            state.eim = item
7149        elseif d == 'l' and state.sim and state.has_r then
7150            head, state = insert_implicit(head, state, outer)
7151        elseif d == 'l' then
7152            state.sim, state.eim, state.has_r = nil, nil, false
7153        end
7154    else
7155        if d == 'an' or d == 'l' then
7156            if nodes[q][3] then  -- nil except after an explicit dir
7157                state.sim = item  -- so we move sim 'inside' the group
7158            else
7159                state.sim = state.sim or item
7160            end
7161            state.eim = item
7162        elseif d == 'r' and state.sim then
7163            head, state = insert_implicit(head, state, outer)
7164        elseif d == 'r' then
7165            state.sim, state.eim = nil, nil
7166        end
7167    end
7168
7169    if isdir then
7170        last = d  -- Don't search back - best save now
7171    elseif d == 'on' and state.san then
7172        state.san = state.san or item
7173        state.ean = item
7174    end
7175
7176    end
7177
7178    return node.prev(head) or head
7179 end
7180 /*basic*/

13 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'},
[0x002B]={c='pr'},
```

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

14 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.
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

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}
\langle /nil \rangle

\newpage

\section{Support for Plain \LaTeX\ (plain.def)}

\subsection{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 \LaTeX\-format. When asked he responded:

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

People can have a file locally hyphen.tex or whatever they like, but they mustn’t diddle with hyphen.tex (or plain.tex except to preload additional fonts).
\end{quote}

The files bplain.tex and bplain.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 \inside, you will get a file called either bplain.fmt or bplain.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 \inside sees, we need to set some category codes just to be able to change the definition of \input.

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

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

\openin 0 hyphen.cfg
\ifeof0
\else
\let\a\input
\fi

201
Then \texttt{\input} is defined to forget about its argument and load \texttt{hyphen.cfg} instead. Once that's done the original meaning of \texttt{\input} can be restored and the definition of \texttt{\a} can be forgotten.

\begin{verbatim}
def\input #1 {% 
  \let\input\a 
  \a\ hyphen.cfg 
  \let\a\undefined 
} 
\fi
\end{verbatim}

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

\begin{verbatim}
⟨bplain⟩\a plain.tex
⟨blplain⟩\a lplain.tex
\end{verbatim}

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

\begin{verbatim}
⟨bplain⟩\def\fmtname{babel-plain}
⟨blplain⟩\def\fmtname{babel-lplain}
\end{verbatim}

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

### 15.2 Emulating some \LaTeX\ X features

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

\begin{verbatim}
⟨⟨∗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}
\end{verbatim}

### 15.3 General tools

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

\begin{verbatim}
\long\def\@firstofone#1{#1} 
\long\def\@firstoftwo#1#2{#1} 
\long\def\@secondoftwo#1#2{#2} 
\def\@nnil{\@nil} 
\def\@gobbletwo#1#2{} 
\def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}} 
\def\@star@or@long#1{\@ifstar{\let\l@ngrel@x\relax#1}{\let\l@ngrel@x\long#1}} 
\let\l@ngrel@x\relax 
\def\@car#1#2\@nil{#1} 
\def\@cdr#1#2\@nil{#2} 
\let\@typeset@protect\relax 
\end{verbatim}

202
LaTeX has the command \@onlypreamble which adds commands to a list of commands that are no longer needed after \begin{document}.

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

Mimick \LaTeX's \texttt{AtBeginDocument}; for this to work the user needs to add \texttt{\begin{document}} to his file.

\def\begindocument{\@begindocumenthook
    \global\let\@begindocumenthook\@undefined
    \do\@preamblecmds
    \global\let\do\noexpand}
\ifx\@begindocumenthook\@undefined
    \def\@begindocumenthook{}
\fi
\@onlypreamble\@begindocumenthook

\def\begindocument{}
\def\AtBeginDocument{\g@addto@macro\begindocumenthook}

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

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

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

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

Mimick \LaTeX's commands to define control sequences.

\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{\@protected@testopt#1\csname\string#1\endcsname{#3}}\expandafter\@yargdef\csname\string#1\endcsname\tw@{#2}{#4}}
\long\def\@yargdef#1#2#3{\@tempcnta#3\relax\advance\@tempcnta\@ne\let\@hash@\relax\edef\reserved@a{\ifx#2\tw@\@hash@1\fi}\@tempcntb#2\@whilenum\@tempcntb<\@tempcnta\do{\edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}}\let\@hash@\@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\}\let\@class@prefix\@empty\let\@class@suffix\@empty\edef\reserved@a\{\string#1\}\long\def\reserved@a\@testopt\reserved@a{\if\reserved@a\relax\let\reserved@a\reserved@a\fi}}
The following little macro \in@ is taken from latex.ltx; it checks whether its first argument is part of its second argument. It uses the boolean \in@; allocating a new boolean inside conditionally executed code is not possible, hence the construct with the temporary definition of \bbl@tempa.

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

\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{} versions; just enough to make things work in plain \TeX{} environments.

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

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

\ifx\bye\@undefined
\advance\count10 by -2\relax
\fi
\ifx\@ifnextchar\@undefined
\let\reserved@d=#1\%
\fi

\def\@ifnextchar#1#2#3(#4){#3 (#4)}

The \LaTeX{} macro \@ifnextchar\@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{} versions; just enough to make things work in plain \TeX{} environments.

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

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

\ifx\bye\@undefined
\advance\count10 by -2\relax
\fi
\ifx\@ifnextchar\@undefined
\let\reserved@d=#1\%
\fi
15.4 Encoding related macros

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

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

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

And a few more “dummy” definitions.
16 Acknowledgements

I would like to thank all who volunteered as $\beta$-testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

References