

# Babel

Localization and  
internationalization

Unicode

T<sub>E</sub>X

pdfT<sub>E</sub>X

LuaT<sub>E</sub>X

XeT<sub>E</sub>X

Version 3.81.2898  
2022/10/22

Javier Bezos  
Current maintainer

Johannes L. Braams  
Original author

# Contents

<b>I</b>	<b>User guide</b>	<b>4</b>
<b>1</b>	<b>The user interface</b>	<b>4</b>
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 . . . . .	26
1.16	Creating a language . . . . .	28
1.17	Digits and counters . . . . .	31
1.18	Dates . . . . .	32
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 . . . . .	41
1.25	Language attributes . . . . .	45
1.26	Hooks . . . . .	45
1.27	Languages supported by babel with ldf files . . . . .	46
1.28	Unicode character properties in luatex . . . . .	48
1.29	Tweaking some features . . . . .	48
1.30	Tips, workarounds, known issues and notes . . . . .	48
1.31	Current and future work . . . . .	49
1.32	Tentative and experimental code . . . . .	50
<b>2</b>	<b>Loading languages with language.dat</b>	<b>50</b>
2.1	Format . . . . .	50
<b>3</b>	<b>The interface between the core of babel and the language definition files</b>	<b>51</b>
3.1	Guidelines for contributed languages . . . . .	52
3.2	Basic macros . . . . .	53
3.3	Skeleton . . . . .	54
3.4	Support for active characters . . . . .	55
3.5	Support for saving macro definitions . . . . .	55
3.6	Support for extending macros . . . . .	55
3.7	Macros common to a number of languages . . . . .	56
3.8	Encoding-dependent strings . . . . .	56
3.9	Executing code based on the selector . . . . .	59
<b>II</b>	<b>Source code</b>	<b>60</b>
<b>4</b>	<b>Identification and loading of required files</b>	<b>60</b>
<b>5</b>	<b>locale directory</b>	<b>60</b>

<b>6</b>	<b>Tools</b>	<b>61</b>
6.1	Multiple languages	65
6.2	The Package File ( <code>\LaTeX</code> , <code>babel.sty</code> )	66
6.3	<code>base</code>	67
6.4	<code>key=value</code> options and other general option	67
6.5	Conditional loading of shorthands	69
6.6	Interlude for Plain	70
<b>7</b>	<b>Multiple languages</b>	<b>70</b>
7.1	Selecting the language	73
7.2	Errors	81
7.3	Hooks	83
7.4	Setting up language files	85
7.5	Shorthands	86
7.6	Language attributes	95
7.7	Support for saving macro definitions	97
7.8	Short tags	98
7.9	Hyphens	98
7.10	Multiencoding strings	100
7.11	Macros common to a number of languages	107
7.12	Making glyphs available	107
7.12.1	Quotation marks	107
7.12.2	Letters	108
7.12.3	Shorthands for quotation marks	109
7.12.4	Umlauts and tremas	110
7.13	Layout	111
7.14	Load engine specific macros	112
7.15	Creating and modifying languages	112
<b>8</b>	<b>Adjusting the Babel behavior</b>	<b>134</b>
8.1	Cross referencing macros	136
8.2	Marks	139
8.3	Preventing clashes with other packages	140
8.3.1	<code>ifthen</code>	140
8.3.2	<code>varioref</code>	140
8.3.3	<code>hhline</code>	141
8.4	Encoding and fonts	142
8.5	Basic bidi support	143
8.6	Local Language Configuration	146
8.7	Language options	147
<b>9</b>	<b>The kernel of Babel (<code>babel.def</code>, <code>common</code>)</b>	<b>150</b>
<b>10</b>	<b>Loading hyphenation patterns</b>	<b>150</b>
<b>11</b>	<b>Font handling with <code>fontspec</code></b>	<b>154</b>
<b>12</b>	<b>Hooks for XeTeX and LuaTeX</b>	<b>157</b>
12.1	XeTeX	157
12.2	Layout	159
12.3	LuaTeX	161
12.4	Southeast Asian scripts	167
12.5	CJK line breaking	168
12.6	Arabic justification	170
12.7	Common stuff	174
12.8	Automatic fonts and ids switching	174
12.9	Bidi	179
12.10	Layout	181
12.11	Lua: transforms	186

12.12	Lua: Auto bidi with basic and basic-r . . . . .	194
<b>13</b>	<b>Data for CJK</b>	<b>204</b>
<b>14</b>	<b>The ‘nil’ language</b>	<b>204</b>
<b>15</b>	<b>Calendars</b>	<b>205</b>
15.1	Islamic . . . . .	206
<b>16</b>	<b>Hebrew</b>	<b>207</b>
<b>17</b>	<b>Persian</b>	<b>211</b>
<b>18</b>	<b>Coptic and Ethiopic</b>	<b>212</b>
<b>19</b>	<b>Buddhist</b>	<b>212</b>
<b>20</b>	<b>Support for Plain T<sub>E</sub>X (plain.def)</b>	<b>213</b>
20.1	Not renaming hyphen.tex . . . . .	213
20.2	Emulating some L <sup>A</sup> T <sub>E</sub> X features . . . . .	214
20.3	General tools . . . . .	214
20.4	Encoding related macros . . . . .	218
<b>21</b>	<b>Acknowledgements</b>	<b>220</b>

## Troubleshootoing

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 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 pdf $\TeX$ , 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 **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 `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:

PDF $\TeX$

```
\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).

LUATEX/XETEX

```

\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  $\TeX$  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`:

```
Package babel Warning: No hyphenation patterns were preloaded for
(babel)                  the language `LANG' into the format.
(babel)                  Please, configure your TeX system to add them and
(babel)                  rebuild the format. Now I will use the patterns
(babel)                  preloaded for \language=0 instead on input line 57.
```

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 (MacTeX, MikTeX, T<sub>E</sub>XLive, 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 (eg, spanish and french).

**EXAMPLE** In L<sup>A</sup>T<sub>E</sub>X, the preamble of the document:

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

would tell L<sup>A</sup>T<sub>E</sub>X 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.

**EXAMPLE** 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}
```

**NOTE** 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 `\language` (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 pdf<sub>tex</sub> follows. The main language is french, which is activated when the document begins. It assumes UTF-8:

PDF<sub>TEX</sub>

```
\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, because the default font supports both languages.

LUATEX/XETEX

```
\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 three-letter word is a valid name for a language (eg. `lu` can be the locale name with tag `khb` or the tag for `lubakatanga`). 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):<sup>1</sup>

```
\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  $\text{\LaTeX}$  (ie, `\usepackage{<language>}`) is deprecated and you will get the error:<sup>2</sup>

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

- Another typical error when using `babel` is the following:<sup>3</sup>

```
! 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.

<sup>1</sup>No predefined “axis” for modifiers are provided because languages and their scripts have quite different needs.

<sup>2</sup>In old versions the error read “You have used an old interface to call `babel`”, not very helpful.

<sup>3</sup>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`  $\{ \langle language \rangle \}$

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 to 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 `\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).

`\foreignlanguage` [*<option-list>*]{*<language>*}{*<text>*}

The command `\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.

**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:

```
\foreignlanguage[date]{polish}{\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 `{\selectlanguage{..} ..}`, which was not always the most convenient way.

## 1.8 Auxiliary language selectors

`\begin{otherlanguage}` {*<language>*} ... `\end{otherlanguage}`

The environment `otherlanguage` does basically the same as `\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:

```
\begingroup
\selectlanguage{<inner-language>}
...
\endgroup
\selectlanguage{<outer-language>}
```

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.

`\begin{otherlanguage*}` [*<option-list>*]{*<language>*} ... `\end{otherlanguage*}`

Same as `\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 `\foreignlanguage`, except when the option `bidi` is set – in this case, `\foreignlanguage` emits a `\leavevmode`, while `otherlanguage*` does not.

## 1.9 More on selection

`\babeltags` {*<tag1>* = *<language1>*, *<tag2>* = *<language2>*, ...}

**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  $\text{\TeX}$  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

```
\babeltags{de = german}
```

you can write

```
text \textde{German text} text
```

and

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

**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 `ruussian`, 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:

```
\foreignlanguage{ruussian}{text \foreignlanguage{polish}{\seename} text}
```

Of course,  $\text{\TeX}$  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:

```
\babelensure{polish}
```

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`.<sup>4</sup> A couple of examples:

```
\babelensure[include=\Today]{spanish}
\babelensure[fontenc=T5]{vietnamese}
```

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,  $\text{\TeX}$  of `\dag`). With `ini` files (see below), captions are ensured by default.

<sup>4</sup>With it, encoded strings may not work as expected.

## 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 `\knbccode`, 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`  $\{ \langle shorthands-list \rangle \}$   
`\shorthandoff`  $* \{ \langle shorthands-list \rangle \}$

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, and an error will be raised otherwise. You can check if a character is a shorthand with `\ifbabelshorthand` (see below).

**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).

**\useshortands** `*{\langle char \rangle}`

The command `\useshortands` 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 `\useshortands*{\langle char \rangle}` 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 `\useshortands`. This restriction will be lifted in a future release.

**\defineshortand** `[\langle language \rangle, \langle language \rangle, ...]{\langle shorthand \rangle}{\langle code \rangle}`

The command `\defineshortand` 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 `\languageshortands{\langle lang \rangle}` to the corresponding `\extras{\langle lang \rangle}`, 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 dictionaries (languages do not define shorthands consistently, and “-”, “\”, “=” have different meanings). You can start with, say:

```
\useshortands*{"}
\defineshortand{"*}{\babelhyphen{soft}}
\defineshortand{"-}{\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:

```
\defineshortand[*polish,*portuguese]{"-}{\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.

**\languageshortands** `{\langle language \rangle}`

The command `\languageshortands` 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).<sup>5</sup> 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{\languageshortands{ngerman}}
```

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

<sup>5</sup>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.

**EXAMPLE** 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]{\{\language shorthands{none}\tipaencoding#1}}
```

**`\babelshorthand`**  $\langle shorthand \rangle$

With this command you can use a shorthand even if (1) not activated in shorthands (in this case only shorthands for the current language are taken into account, ie, 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.)

**EXAMPLE** 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:<sup>6</sup>

**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** `

**Kurmanji** ^

**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.<sup>7</sup>

**`\ifbabelshorthand`**  $\langle character \rangle \{ \langle true \rangle \} \{ \langle false \rangle \}$

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

**`\aliasshorthand`**  $\langle original \rangle \{ \langle alias \rangle \}$

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

<sup>6</sup>Thanks to Enrico Gregorio

<sup>7</sup>This declaration serves to nothing, but it is preserved for backward compatibility.

character / over " in typing Polish texts, this can be achieved by entering `\aliasshorthand{"}{/}`. 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, `\aliasshorthands` is ignored).

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

```
\aliasshorthand{~}{^}
\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 is 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  $\TeX$  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  $\TeX$  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  $\epsilon\TeX$  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.

**nocase** **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.<sup>8</sup>

**hyphenmap=** `off` | `first` | `select` | `other` | `other*`

**New 3.9g** Sets the behavior of case mapping for hyphenation, provided the language defines it.<sup>9</sup> It can take the following values:

**off** deactivates 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.<sup>10</sup>

**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.<sup>11</sup>

**bidi=** `default` | `basic` | `basic-r` | `bidi-l` | `bidi-r`

**New 3.14** Selects the bidi algorithm to be used in `luatex` and `xetex`. See sec. 1.24.

**layout=**

**New 3.16** Selects which layout elements are adapted in bidi documents. See sec. 1.24.

**provide=** \*

---

<sup>8</sup>You can use alternatively the package `silence`.

<sup>9</sup>Turned off in plain.

<sup>10</sup>Duplicated options count as several ones.

<sup>11</sup>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.

**New 3.49** An alternative to `\babelprovide` for languages passed as options. See section 1.13, which describes also the variants `provide+=` and `provide*=`.

## 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`  $\langle\textit{option-name}\rangle\{\langle\textit{code}\rangle\}$

This command is currently the only provided by `base`. Executes  $\langle\textit{code}\rangle$  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  $\langle\textit{option-name}\rangle$  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  $\text{\TeX}$ , 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 have been devised as a resource for other packages. To easy interoperability between  $\text{\TeX}$  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 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 have 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. Khemer clusters are rendered wrongly with the default renderer. The comment about Indic scripts and lualatex also applies here. Some quick patterns can help, with something similar to:

```
\babelprovide[import, hyphenrules=+]{lao}
\babelpatterns[lao]{lᦺ lᦴ lᦶ lᦸ lᦺ lᦴ} % Random
```

**East Asia scripts** Settings for either Simplified or 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 `ldf` 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 hyphenation 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):

---

af	Afrikaans <sup>ul</sup>	be	Belarusian <sup>ul</sup>
agq	Aghem	bem	Bemba
ak	Akan	bez	Bena
am	Amharic <sup>ul</sup>	bg	Bulgarian <sup>ul</sup>
ar	Arabic <sup>ul</sup>	bm	Bambara
ar-DZ	Arabic <sup>ul</sup>	bn	Bangla <sup>ul</sup>
ar-EG	Arabic <sup>ul</sup>	bo	Tibetan <sup>u</sup>
ar-IQ	Arabic <sup>ul</sup>	brx	Bodo
ar-JO	Arabic <sup>ul</sup>	bs-Cyrl	Bosnian
ar-LB	Arabic <sup>ul</sup>	bs-Latn	Bosnian <sup>ul</sup>
ar-MA	Arabic <sup>ul</sup>	bs	Bosnian <sup>ul</sup>
ar-PS	Arabic <sup>ul</sup>	ca	Catalan <sup>ul</sup>
ar-SA	Arabic <sup>ul</sup>	ce	Chechen
ar-SY	Arabic <sup>ul</sup>	cgg	Chiga
ar-TN	Arabic <sup>ul</sup>	chr	Cherokee
as	Assamese	ckb	Central Kurdish
asa	Asu	cop	Coptic
ast	Asturian <sup>ul</sup>	cs	Czech <sup>ul</sup>
az-Cyrl	Azerbaijani	cu	Church Slavic
az-Latn	Azerbaijani	cu-Cyrs	Church Slavic
az	Azerbaijani <sup>ul</sup>	cu-Glag	Church Slavic
bas	Basaa	cy	Welsh <sup>ul</sup>

da	Danish <sup>ul</sup>	ig	Igbo
dav	Taita	ii	Sichuan Yi
de-AT	German <sup>ul</sup>	is	Icelandic <sup>ul</sup>
de-CH	Swiss High German <sup>ul</sup>	it	Italian <sup>ul</sup>
de	German <sup>ul</sup>	ja	Japanese <sup>u</sup>
dje	Zarma	jgo	Ngomba
dsb	Lower Sorbian <sup>ul</sup>	jmc	Machame
dua	Duala	ka	Georgian <sup>ul</sup>
dyo	Jola-Fonyi	kab	Kabyle
dz	Dzongkha	kam	Kamba
ebu	Embu	kde	Makonde
ee	Ewe	kea	Kabuverdianu
el	Greek <sup>ul</sup>	khq	Koyra Chiini
el-polyton	Polytonic Greek <sup>ul</sup>	ki	Kikuyu
en-AU	English <sup>ul</sup>	kk	Kazakh
en-CA	English <sup>ul</sup>	kkj	Kako
en-GB	English <sup>ul</sup>	kl	Kalaallisut
en-NZ	English <sup>ul</sup>	kln	Kalenjin
en-US	English <sup>ul</sup>	km	Khmer
en	English <sup>ul</sup>	kmr	Northern Kurdish <sup>u</sup>
eo	Esperanto <sup>ul</sup>	kn	Kannada <sup>ul</sup>
es-MX	Spanish <sup>ul</sup>	ko	Korean <sup>u</sup>
es	Spanish <sup>ul</sup>	kok	Konkani
et	Estonian <sup>ul</sup>	ks	Kashmiri
eu	Basque <sup>ul</sup>	ksb	Shambala
ewo	Ewondo	ksf	Bafia
fa	Persian <sup>ul</sup>	ksh	Colognian
ff	Fulah	kw	Cornish
fi	Finnish <sup>ul</sup>	ky	Kyrgyz
fil	Filipino	lag	Langi
fo	Faroese	lb	Luxembourgish <sup>ul</sup>
fr	French <sup>ul</sup>	lg	Ganda
fr-BE	French <sup>ul</sup>	lkt	Lakota
fr-CA	French <sup>ul</sup>	ln	Lingala
fr-CH	French <sup>ul</sup>	lo	Lao <sup>ul</sup>
fr-LU	French <sup>ul</sup>	lrc	Northern Luri
fur	Friulian <sup>ul</sup>	lt	Lithuanian <sup>ul</sup>
fy	Western Frisian	lu	Luba-Katanga
ga	Irish <sup>ul</sup>	luo	Luo
gd	Scottish Gaelic <sup>ul</sup>	luy	Luyia
gl	Galician <sup>ul</sup>	lv	Latvian <sup>ul</sup>
grc	Ancient Greek <sup>ul</sup>	mas	Masai
gsw	Swiss German	mer	Meru
gu	Gujarati	mfe	Morisyen
guz	Gusii	mg	Malagasy
gv	Manx	mgh	Makhuwa-Meetto
ha-GH	Hausa	mgo	Meta'
ha-NE	Hausa <sup>l</sup>	mk	Macedonian <sup>ul</sup>
ha	Hausa	ml	Malayalam <sup>ul</sup>
haw	Hawaiian	mn	Mongolian
he	Hebrew <sup>ul</sup>	mr	Marathi <sup>ul</sup>
hi	Hindi <sup>u</sup>	ms-BN	Malay <sup>l</sup>
hr	Croatian <sup>ul</sup>	ms-SG	Malay <sup>l</sup>
hsb	Upper Sorbian <sup>ul</sup>	ms	Malay <sup>ul</sup>
hu	Hungarian <sup>ul</sup>	mt	Maltese
hy	Armenian <sup>u</sup>	mua	Mundang
ia	Interlingua <sup>ul</sup>	my	Burmese
id	Indonesian <sup>ul</sup>	mzn	Mazanderani

naq	Nama	sn	Shona
nb	Norwegian Bokmål <sup>ul</sup>	so	Somali
nd	North Ndebele	sq	Albanian <sup>ul</sup>
ne	Nepali	sr-Cyrl-BA	Serbian <sup>ul</sup>
nl	Dutch <sup>ul</sup>	sr-Cyrl-ME	Serbian <sup>ul</sup>
nmg	Kwasio	sr-Cyrl-XK	Serbian <sup>ul</sup>
nn	Norwegian Nynorsk <sup>ul</sup>	sr-Cyrl	Serbian <sup>ul</sup>
nnh	Ngiemboon	sr-Latn-BA	Serbian <sup>ul</sup>
no	Norwegian	sr-Latn-ME	Serbian <sup>ul</sup>
nus	Nuer	sr-Latn-XK	Serbian <sup>ul</sup>
nyn	Nyankole	sr-Latn	Serbian <sup>ul</sup>
om	Oromo	sr	Serbian <sup>ul</sup>
or	Odia	sv	Swedish <sup>ul</sup>
os	Ossetic	sw	Swahili
pa-Arab	Punjabi	ta	Tamil <sup>u</sup>
pa-Guru	Punjabi	te	Telugu <sup>ul</sup>
pa	Punjabi	teo	Teso
pl	Polish <sup>ul</sup>	th	Thai <sup>ul</sup>
pms	Piedmontese <sup>ul</sup>	ti	Tigrinya
ps	Pashto	tk	Turkmen <sup>ul</sup>
pt-BR	Portuguese <sup>ul</sup>	to	Tongan
pt-PT	Portuguese <sup>ul</sup>	tr	Turkish <sup>ul</sup>
pt	Portuguese <sup>ul</sup>	twq	Tasawaq
qu	Quechua	tzm	Central Atlas Tamazight
rm	Romansh <sup>ul</sup>	ug	Uyghur
rn	Rundi	uk	Ukrainian <sup>ul</sup>
ro	Romanian <sup>ul</sup>	ur	Urdu <sup>ul</sup>
ro-MD	Moldavian <sup>ul</sup>	uz-Arab	Uzbek
rof	Rombo	uz-Cyrl	Uzbek
ru	Russian <sup>ul</sup>	uz-Latn	Uzbek
rw	Kinyarwanda	uz	Uzbek
rwk	Rwa	vai-Latn	Vai
sa-Beng	Sanskrit	vai-Vaii	Vai
sa-Deva	Sanskrit	vai	Vai
sa-Gujr	Sanskrit	vi	Vietnamese <sup>ul</sup>
sa-Knda	Sanskrit	vun	Vunjo
sa-Mlym	Sanskrit	wae	Walser
sa-Telu	Sanskrit	xog	Soga
sa	Sanskrit	yav	Yangben
sah	Sakha	yi	Yiddish
saq	Samburu	yo	Yoruba
sbp	Sangu	yue	Cantonese
se	Northern Sami <sup>ul</sup>	zgh	Standard Moroccan Tamazight
seh	Sena		
ses	Koyraboro Senni	zh-Hans-HK	Chinese <sup>u</sup>
sg	Sango	zh-Hans-MO	Chinese <sup>u</sup>
shi-Latn	Tachelhit	zh-Hans-SG	Chinese <sup>u</sup>
shi-Tfng	Tachelhit	zh-Hans	Chinese <sup>u</sup>
shi	Tachelhit	zh-Hant-HK	Chinese <sup>u</sup>
si	Sinhala	zh-Hant-MO	Chinese <sup>u</sup>
sk	Slovak <sup>ul</sup>	zh-Hant	Chinese <sup>u</sup>
sl	Slovenian <sup>ul</sup>	zh	Chinese <sup>u</sup>
smn	Inari Sami	zu	Zulu

In some contexts (currently `\babel font`) an ini file may be loaded by its name. Here is the list of the names currently supported. With these languages, `\babel font` 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 \babelprovide with a valueless import.

---

aghem	chinese-hans-mo
akan	chinese-hans-sg
albanian	chinese-hans
american	chinese-hant-hk
amharic	chinese-hant-mo
ancientgreek	chinese-hant
arabic	chinese-simplified-hongkongsarchina
arabic-algeria	chinese-simplified-macausarchina
arabic-DZ	chinese-simplified-singapore
arabic-morocco	chinese-simplified
arabic-MA	chinese-traditional-hongkongsarchina
arabic-syria	chinese-traditional-macausarchina
arabic-SY	chinese-traditional
armenian	chinese
assamese	churchslavic
asturian	churchslavic-cyrs
asu	churchslavic-oldcyrillic <sup>12</sup>
australian	churchsslavic-glag
austrian	churchsslavic-glagolitic
azerbaijani-cyrillic	cognian
azerbaijani-cyrl	cornish
azerbaijani-latin	croatian
azerbaijani-latn	czech
azerbaijani	danish
bafia	duala
bambara	dutch
basaa	dzongkha
basque	embu
belarusian	english-au
bemba	english-australia
bena	english-ca
bangla	english-canada
bodo	english-gb
bosnian-cyrillic	english-newzealand
bosnian-cyrl	english-nz
bosnian-latin	english-unitedkingdom
bosnian-latn	english-unitedstates
bosnian	english-us
brazilian	english
breton	esperanto
british	estonian
bulgarian	ewe
burmese	ewondo
canadian	faroeese
cantonese	filipino
catalan	finnish
centralatlastamazight	french-be
centralkurdish	french-belgium
chechen	french-ca
cherokee	french-canada
chiga	french-ch
chinese-hans-hk	french-lu

<sup>12</sup>The name in the CLDR is Old Church Slavonic Cyrillic, but it has been shortened for practical reasons.

french-luxembourg	lowersorbian
french-switzerland	lsorbian
french	lubakatanga
friulian	luo
fulah	luxembourgish
galician	luyia
ganda	macedonian
georgian	machame
german-at	makhuwameetto
german-austria	makonde
german-ch	malagasy
german-switzerland	malay-bn
german	malay-brunei
greek	malay-sg
gujarati	malay-singapore
gusii	malay
hausa-gh	malayalam
hausa-ghana	maltese
hausa-ne	manx
hausa-niger	marathi
hausa	masai
hawaiian	mazanderani
hebrew	meru
hindi	meta
hungarian	mexican
icelandic	mongolian
igbo	morisyen
inarisami	mundang
indonesian	nama
interlingua	nepali
irish	newzealand
italian	ngiemboon
japanese	ngomba
jolafonyi	norsk
kabuverdianu	northernluri
kabyle	northernsami
kako	northndebele
kalaallisut	norwegianbokmal
kalenjin	norwegiannynorsk
kamba	nswissgerman
kannada	nuer
kashmiri	nyankole
kazakh	nynorsk
khmer	occitan
kikuyu	oriya
kinyarwanda	oromo
konkani	ossetic
korean	pashto
koyraborosenni	persian
koyrachiini	piedmontese
kwasio	polish
kyrgyz	polytonicgreek
lakota	portuguese-br
langi	portuguese-brazil
lao	portuguese-portugal
latvian	portuguese-pt
lingala	portuguese
lithuanian	punjabi-arab



punjabi-arabic	soga
punjabi-gurmukhi	somali
punjabi-guru	spanish-mexico
punjabi	spanish-mx
quechua	spanish
romanian	standardmoroccantamazight
romansh	swahili
rombo	swedish
rundi	swissgerman
russian	tachelhit-latin
rwa	tachelhit-latn
sakha	tachelhit-tfng
samburu	tachelhit-tifinagh
samin	tachelhit
sango	taita
sangu	tamil
sanskrit-beng	tasawaq
sanskrit-bengali	telugu
sanskrit-deva	teso
sanskrit-devanagari	thai
sanskrit-gujarati	tibetan
sanskrit-gujr	tigrinya
sanskrit-kannada	tongan
sanskrit-knda	turkish
sanskrit-malayalam	turkmen
sanskrit-mlym	ukenglish
sanskrit-telu	ukrainian
sanskrit-telugu	uppersorbian
sanskrit	urdu
scottishgaelic	usenglish
sena	usorbian
serbian-cyrillic-bosniaherzegovina	uyghur
serbian-cyrillic-kosovo	uzbek-arab
serbian-cyrillic-montenegro	uzbek-arabic
serbian-cyrillic	uzbek-cyrillic
serbian-cyrl-ba	uzbek-cyrl
serbian-cyrl-me	uzbek-latin
serbian-cyrl-xk	uzbek-latn
serbian-cyrl	uzbek
serbian-latin-bosniaherzegovina	vai-latin
serbian-latin-kosovo	vai-latn
serbian-latin-montenegro	vai-vai
serbian-latin	vai-vaii
serbian-latn-ba	vai
serbian-latn-me	vietnam
serbian-latn-xk	vietnamese
serbian-latn	vunjo
serbian	walser
shambala	welsh
shona	westernfrisian
sichuanyi	yangben
sinhala	yiddish
slovak	yoruba
slovene	zarma
slovenian	zulu afrikaans

---

### 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=abcdefghijkl`. 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

**New 3.15** Babel provides a high level interface on top of `fontspec` to select fonts. There is no need to load `fontspec` explicitly – babel does it for you with the first `\babelfont`.<sup>13</sup>

**`\babelfont`** [*<language-list>*]{*<font-family>*}[*<font-options>*]{*<font-name>*}

**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.

LUATEX/XETEX

```
\documentclass{article}

\usepackage[swedish, bidi=default]{babel}

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עִבְרִית} svenska.

\end{document}
```

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

LUATEX/XETEX

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

<sup>13</sup>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 `rm`, `sf` or `tt`. This is the preferred way to select fonts in addition to the three basic families.

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

LUATEX/XETEX

```
\babelfont{kai}{FandolKai}
```

Now, `\kaifamily` and `\kaidefault`, as well as `\textkai` are at your disposal.

**NOTE** You may load `fontspec` explicitly. For example:

LUATEX/XETEX

```
\usepackage{fontspec}
\newfontscript{Devanagari}{deva}
\babelfont[hindi]{rm}{Shobhika}
```

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 `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 `Script` when declaring a font with `\babelfont` (nor `Language`). In fact, it is even discouraged.

**NOTE** `\fontspec` is not touched at all, only the preset font families (`rm`, `sf`, `tt`, and the like). If a language is switched when an *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 `Language` and `Script` just pass these values to the *font*, and do *not* set the script for the *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 `\setxxxxfont` and `\babelfont` at the same time is discouraged, but very often works as expected. However, be aware with `\setxxxxfont` the language system will not be set by `babel` and should be set with `fontspec` if necessary.

**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 `\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:

`\setlocalecaption`  $\{\langle language-name \rangle\}\{\langle caption-name \rangle\}\{\langle string \rangle\}$

**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:

```
\setlocalecaption{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.licr` one.)

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

```
\addto\captionseenglish{%  
  \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\extrarussian{\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:

```
\usepackage[danish]{babel}  
\babelprovide[captions=da, hyphenrules=nohyphenation]{danish}
```

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 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).

`\babelprovide` [*options*]{*language-name*}

If the language *language-name* has not been loaded as class or package option and there are no *options*, 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`, *language-name* 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:

```
Package babel Warning: \chaptername not set for 'mylang'. Please,
(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:

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

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

```
\babelprovide[import=en-US]{enUS}
```

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 `\selectlanguage{arhinish}` 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=` *language-tag*

**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, and that is often the recommended option. 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 is best written as:

```
\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, which prints the date for the current locale.

**captions=** <language-tag>

Loads only the strings. For example:

```
\babelprovide[captions=hu]{hungarian}
```

**hyphenrules=** <language-list>

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 hyphenrules 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 T<sub>E</sub>X 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 \babelpatterns, as for example:

```
\babelprovide[hyphenrules=+]{neo}  
\babelpatterns[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 breking 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:

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

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

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

Remember there is an alternative syntax for the latter:

```
\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=**  $\langle script-name \rangle$

**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=**  $\langle language-name \rangle$

**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=**  $\langle counter-name \rangle$

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

**Alph=**  $\langle counter-name \rangle$

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 | letters`

**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`). Characters can be added or modified with `\babelcharproperty`.

**New 3.81** Option `letters` restricts the ‘actions’ to letters, in the  $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  sense (i. e., with `catcode 11`). Digits and punctuation are then considered part of current locale (as set by a selector).

**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.

**intraspace=**  $\langle base \rangle \langle shrink \rangle \langle stretch \rangle$

Sets the interword space for the writing system of the language, in em units (so, `0 .1 0` is `0em` plus `.1em`). Like `\spaceskip`, the em unit applied is that of the current text (more precisely, the previous glyph). Currently used only in Southeast Asian scripts, like Thai, and CJK.

**intrapenalty=**  $\langle\textit{penalty}\rangle$

Sets the interword penalty for the writing system of this language. Currently used only in Southeast Asian scripts, like Thai. Ignored if 0 (which is the default value).

**transforms=**  $\langle\textit{transform-list}\rangle$

See section 1.21.

**justification=** kashida | elongated | unhyphenated | padding

**New 3.59** There are currently three options, mainly for the Arabic script. It sets the linebreaking and justification method, which can be based on the the ARABIC TATWEEL character or in the ‘justification alternatives’ OpenType table (jalt). For an explanation see the [babel site](#).

**New 3.81** The option padding has been devised primarily for Tibetan. It’s still somewhat experimental. Again, there is an explanation in the [babel site](#).

**linebreaking=** **New 3.59** Just a synonymous for justification.

**NOTE** (1) If you need shorthands, you can define them with `\usesshorthands` and `\defineshorthand` as described above. (2) Captions and `\today` are “ensured” with `\babelensure` (this is the default in ini-based languages).

## 1.17 Digits and counters

**New 3.20** About thirty ini files define a field named `digits.native`. When it is present, two macros are created: `\<language>digits` and `\<language>counter` (only xetex and luatex). With the first, a string of ‘Latin’ digits are converted to the native digits of that language; the second takes a counter name as argument. With the option `maparabic` in `\babelprovide`, `\arabic` is redefined to produce the native digits (this is done *globally*, to avoid inconsistencies in, for example, page numbering, and note as well dates do not rely on `\arabic`.)

For example:

```
\babelprovide[import]{telugu}
% Or also, if you want:
% \babelprovide[import, maparabic]{telugu}
\babelfont{rm}{Gautami} % With luatex, better with Harfbuzz
\begin{document}
\telugudigits{1234}
\telugucounter{section}
\end{document}
```

Languages providing native digits in all or some variants are:

Arabic	Persian	Lao	Odia	Urdu
Assamese	Gujarati	Northern Luri	Punjabi	Uzbek
Bangla	Hindi	Malayalam	Pashto	Vai
Tibetar	Khmer	Marathi	Tamil	Cantonese
Bodo	Kannada	Burmese	Telugu	Chinese
Central Kurdish	Konkani	Mazanderani	Thai	
Dzongkha	Kashmiri	Nepali	Uyghur	

**New 3.30** 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 (ie, to the node list as generated by the T<sub>E</sub>X 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.



```
\localenumeral {\style}{\number}
\localecounterl {\style}{\counter}
```

**New 3.41** Many ‘ini’ locale files has 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 \alph and \Alph print. For example:

```
\babelprovide[alph=alphabetic]{thai}
```

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, yemsa  
**Arabic** abjad, maghrebi.abjad  
**Armenian** lower.letter, upper.letter  
**Belarusan, Bulgarian, Church Slavic, Macedonian, Serbian** lower, upper  
**Bangla** alphabetic  
**Central Kurdish** alphabetic  
**Chinese** cjk-earthly-branch, cjk-heavenly-stem, circled.ideograph, parenthesized.ideograph, fullwidth.lower.alpha, fullwidth.upper.alpha  
**Church Slavic (Glagolitic)** letters  
**Coptic** epact, lower.letters  
**French** date.day (mainly for internal use).  
**Georgian** letters  
**Greek** lower.modern, upper.modern, lower.ancient, upper.ancient (all with keraia)  
**Hebrew** letters (neither geresh nor gershayim yet)  
**Hindi** alphabetic  
**Italian** lower.legal, upper.legal  
**Japanese** hiragana, hiragana.iroha, katakana, katakana.iroha, circled.katakana, informal, formal, cjk-earthly-branch, cjk-heavenly-stem, circled.ideograph, parenthesized.ideograph, fullwidth.lower.alpha, fullwidth.upper.alpha  
**Khmer** consonant  
**Korean** consonant, syllabe, hanja.informal, hanja.formal, hangul.formal, cjk-earthly-branch, cjk-heavenly-stem, circled.ideograph, parenthesized.ideograph, fullwidth.lower.alpha, fullwidth.upper.alpha  
**Marathi** alphabetic  
**Persian** abjad, alphabetic  
**Russian** lower, lower.full, upper, upper.full  
**Syriac** letters  
**Tamil** ancient  
**Thai** alphabetic  
**Ukrainian** lower, lower.full, upper, upper.full

**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** [*<calendar=..., variant=..., convert>*]{*<year>*}{*<month>*}{*<day>*}

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 and calendar=coptic). However, with the option convert it's converted (using internally the following command).

Even with a certain calendar there may be variants. In Kurmanji the default variant prints something like 30. *Çîleya Pêşîn 2019*, but with variant=iza fa it prints 31'ê *Çîleya Pêşînê 2019*.

**\babelcalendar** [*<date>*]{*<calendar>*}{*<year-macro>*}{*<month-macro>*}{*<day-macro>*}

**New 3.76** Although calendars aren't the primary concern of babel, the package should be able to, at least, generate correctly the current date in the way users would expect in their own culture. Currently, \localedate can print dates in a few calendars (provided the ini locale file has been imported), but year, month and day had to be entered by hand, which is very inconvenient. With this macro, the current date is converted and stored in the three last arguments, which must be macros: allowed calendars are buddhist, coptic, hebrew, islamic-civil, islamic-umalqura, persian. The optional argument converts the given date, in the form '*<year>*-'*<month>*-'*<day>*'. Please, refer to the page on the news for 3.76 in the babel site for further details.

## 1.19 Accessing language info

**\language** The control sequence \language 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** {*<language>*}{*<true>*}{*<false>*}

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 T<sub>E</sub>X 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** \*{*<field>*}

**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). This is the value to be used for the 'real' provided tag (babel may fill other fields if they are considered necessary).

language.tag.bcp47 is the BCP 47 language tag.

tag.opentype 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. This is a required field for the fonts to be correctly set up, and therefore it should be always defined.

script.tag.opentype is the tag used by OpenType (usually, but not always, the same as BCP 47).

region.tag.bcp47 is the BCP 47 tag of the region or territory. Defined only if the locale loaded actually contains it (eg, es-MX does, but es doesn't), which is how locales behave in the CLDR. **New 3.75**

`variant.tag.bcp47` is the BCP 47 tag of the variant (in the BCP 47 sense, like 1901 for German). **New 3.75**

`extension.<s>.tag.bcp47` is the BCP 47 value of the extension whose singleton is `<s>` (currently the recognized singletons are x, t and u). The internal syntax can be somewhat complex, and this feature is still somewhat tentative. An example is `classicalatin` which sets `extension.x.tag.bcp47` to `classic`. **New 3.75**

**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.

**New 3.75** Sometimes, it comes in handy to be able to use `\localeinfo` in an expandable way even if something went wrong (for example, the locale currently active is undefined). For these cases, `localeinfo*` just returns an empty string instead of raising an error. Bear in mind that `babel`, following the CLDR, may leave the region unset, which means `\getlanguageproperty*`, described below, is the preferred command, so that the existence of a field can be checked before. This also means building a string with the language and the region with `\localeinfo*{language.tab.bcp47}-\localeinfo*{region.tab.bcp47}` is not usually a good idea (because of the hyphen).

**\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 פרק.

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.

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

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 stored 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 (when it makes sense) as an attribute, too.

**\LocaleForEach** `{\code}`

`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.

**ensureinfo=off** **New 3.75** Previously, ini files were loaded only with `\babelprovide` and also when languages are selected if there is a `\babelfont` or they have not been explicitly declared. Now the ini files are loaded (and therefore the corresponding data) even if these two conditions are not met (in previous versions you had to enable it with `\BabelEnsureInfo` in the preamble). Because of the way this feature works, problems are very unlikely, but there is a switch as a package option to turn the new behavior off (`ensureinfo=off`).

## 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 T<sub>E</sub>X 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 T<sub>E</sub>X 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 T<sub>E</sub>X, - 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.
- `\babelhyphen{nobreak}` inserts a hard hyphen without a break after it (even if a space follows).
- `\babelhyphen{empty}` inserts a break opportunity without a hyphen at all.
- `\babelhyphen{\text}` is a hard “hyphen” using `\text` instead. A typical case is `\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: `\babelhyphen*{soft}` (which in most cases is equivalent to the original \-), `\babelhyphen*{hard}`, etc.

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

There are also some differences with L<sup>A</sup>T<sub>E</sub>X: (1) the character used is that set for the current font, while in L<sup>A</sup>T<sub>E</sub>X it is hardwired to - (a typical value); (2) the hyphen to be used in fonts with a negative `\hyphenchar` is -, like in L<sup>A</sup>T<sub>E</sub>X, but it can be changed to another value by redefining `\babeinullhyphen`; (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).

`\babelhyphenation` [`\language`], [`\language`], ...] `{\exceptions}`

**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). Multiple declarations work much like `\hyphenation` (last wins), but 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 `\lccodes`’s done in `\extras{lang}` as well as the language-specific encoding (not set in the preamble by default). Multiple `\babelhyphenation`’s are allowed. For example:

```
\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.

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

**NOTE** Use `\babelhyphenation` instead of `\hyphenation` to set hyphenation exceptions in the preamble before any language is explicitly set with a selector. In the preamble the hyphenation rules are not always fully set up and an error can be raised.

`\begin{hyphenrules}`  $\langle\langle\text{language}\rangle\rangle$  ... `\end{hyphenrules}`

The environment `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 `language.dat` the ‘language’ nohyphenation is defined by loading `zerohyph.tex`. It deactivates language shorthands, too (but not user shorthands). Except for these simple uses, `hyphenrules` is deprecated and other `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`).

`\babelpatterns` [ $\langle\langle\text{language}\rangle\rangle$ ,  $\langle\langle\text{language}\rangle\rangle$ , ...]  $\langle\langle\text{patterns}\rangle\rangle$

**New 3.9m** In `luatex` only,<sup>14</sup> adds or replaces patterns for the languages given or, without the optional argument, for *all* languages. If a pattern for a certain combination already exists, it gets replaced by the new one.

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<lang>` as well as the language-specific encoding (not set in the preamble by default). Multiple `\babelpatterns`’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 `\babelprovide` 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 Khemer is activated automatically if a language with one of those scripts are loaded with `\babelprovide`. 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.<sup>15</sup>

It currently embraces `\babelprehyphenation` and `\babelposthyphenation`.

**New 3.57** Several ini files predefine some transforms. They are activated with the key `transforms` in `\babelprovide`, 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:

<sup>14</sup>With `luatex` exceptions and patterns can be modified almost freely. However, this is very likely a task for a separate package and `babel` only provides the most basic tools.

<sup>15</sup>They are similar in concept, but not the same, as those in Unicode. The main inspiration for this feature is the Omega transformation processes.

```
\usepackage[magyar]{babel}
\babelprovide[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. (A few may still require some fine-tuning. More to follow in future releases.)

Arabic	<code>transliteration.dad</code>	Applies the transliteration system devised by Yannis Haralambous for dad (simple and $\TeX$ -friendly). Not yet complete, but sufficient for most texts.
Croatian	<code>digraphs.ligatures</code>	Ligatures <i>DŽ, Dž, dž, LJ, Lj, lj, NJ, Nj, nj</i> . 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.
Czech, Polish, Portuguese, Slovak, Spanish	<code>hyphen.repeat</code>	Explicit hyphens behave like <code>\babelhyphen{repeat}</code> .
Czech, Polish, Slovak	<code>oneletter.nobreak</code>	Converts a space after a non-syllabic preposition or conjunction into a non-breaking space.
Finnish	<code>prehyphen.nobreak</code>	Line breaks just after hyphens prepended to words are prevented, like in “pakastekaapit ja -arkut”.
Greek	<code>diaeresis.hyphen</code>	Removes the diaeresis above iota and upsilon if hyphenated just before. It works with the three variants.
Greek	<code>transliteration.omega</code>	Although the provided combinations are not the full set, this transform follows the syntax of Omega: $\circ$ for the circumflex, $\upsilon$ for digamma, and so on. For better compatibility with Levy’s system, $\sim$ (as ‘string’) is an alternative to $\circ$ . $\acute{\phantom{a}}$ is tonos in Monotonic Greek, but oxia in Polytonic and Ancient Greek.
Greek	<code>sigma.final</code>	The transliteration system above does not convert the sigma at the end of a word (on purpose). This transform does it. To prevent the conversion (an abbreviation, for example), write “s.”
Hindi, Sanskrit	<code>transliteration.hk</code>	The Harvard-Kyoto system to romanize Devanagari.
Hindi, Sanskrit	<code>punctuation.space</code>	Inserts a space before the following four characters: <code>!?:;</code> .

Hungarian	<code>digraphs.hyphen</code>	Hyphenates the long digraphs <i>ccs</i> , <i>ddz</i> , <i>ggy</i> , <i>lly</i> , <i>nnny</i> , <i>ssz</i> , <i>tty</i> and <i>zzs</i> as <i>cs-cs</i> , <i>dz-dz</i> , etc.
Indic scripts	<code>danda.nobreak</code>	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.
Latin	<code>digraphs.ligatures</code>	Replaces the groups <i>ae</i> , <i>AE</i> , <i>oe</i> , <i>OE</i> with <i>æ</i> , <i>Æ</i> , <i>œ</i> , <i>Œ</i> .
Latin	<code>letters.noj</code>	Replaces <i>j</i> , <i>J</i> with <i>i</i> , <i>I</i> .
Latin	<code>letters.uv</code>	Replaces <i>v</i> , <i>U</i> with <i>u</i> , <i>V</i> .
Sanskrit	<code>transliteration.iast</code>	The IAST system to romanize Devanagari. <sup>16</sup>
Serbian	<code>transliteration.gajica</code>	(Note <i>serbian</i> with <i>ini</i> files refers to the Cyrillic script, which is here the target.) The standard system devised by Ljudevit Gaj.
Arabic, Persian	<code>kashida.plain</code>	Experimental. A very simple and basic transform for ‘plain’ Arabic fonts, which attempts to distribute the <i>tatwil</i> as evenly as possible (starting at the end of the line). See the news for version 3.59.

**`\babelposthyphenation`** [*⟨options⟩*]{*⟨hyphenrules-name⟩*}{*⟨lua-pattern⟩*}{*⟨replacement⟩*}

**New 3.37-3.39** With *luatex* it is possible to define non-standard hyphenation rules, like *f-f* → *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):

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

In the replacements, a captured char may be mapped to another, too. For example, if the first capture reads *([ıú])*, the replacement could be *{1|ıú|ıú}*, which maps *ı* to *ı*, and *ú* to *ú*, so that the diaeresis is removed.

This feature is activated with the first `\babelposthyphenation` or `\babelprehyphenation`.

**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 `\newattribute`, `\setattribute` and `\unsetattribute`. The following example shows how to use it, provided an attribute named `\latinnoj` has been declared:

```
\babelprehyphenation[attribute=\latinnoj]{latin}{ J }{ string = I }
```

See the [babel site](#) for a more detailed description and some examples. It also describes a few additional replacement types (*string*, *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*.



`\babelprehyphenation` [*options*]{*locale-name*}{*lua-pattern*}{*replacement*}

**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 | 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 `\babelposthyphenation` or `\babelprehyphenation`.

**EXAMPLE** You can replace a character (or series of them) by another character (or series of them). Thus, to enter *ž* as *zh* and *š* as *sh* in a newly created locale for transliterated Russian:

```
\babelprovide[hyphenrules=+]{russian-latin} % Create locale
\babelprehyphenation{russian-latin}{([sz])h} % Create rule
{
  string = {1|sz|šž},
  remove
}
```

**EXAMPLE** The following rule prevent the word “a” from being at the end of a line:

```
\babelprehyphenation{english}{|a|}
{ }, { }, % Keep first space and a
{ insert, penalty = 10000 }, % Insert penalty
{ } % Keep last space
}
```

**NOTE** With *luatex* there is another approach to make text transformations, with the function `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 `\babel font`. The *transforms* mechanism supplements rather than replaces OTF features.

With *xetex*, where *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: *fr-Latn-FR* → *fr-Latn* → *fr-FR* → *fr*. Languages with the same resolved name are considered the same. Case is normalized before, so that *fr-latn-fr* → *fr-Latn-FR*. If a tag and a name overlap, the tag takes precedence.

Here is a minimal example:

```
\documentclass{article}

\usepackage[danish]{babel}
```



```

\babeladjust{
  autoload.bcp47 = on,
  autoload.bcp47.options = import
}

\begin{document}

Chapter in Danish: \chaptername.

\selectlanguage{de-AT}

\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 `\babeladjust` with the following parameters:

`autoload.bcp47` with values `on` and `off`.

`autoload.bcp47.options`, which are passed to `\babelprovide`; empty by default, but you may add `import` (features defined in the corresponding `babel-...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.<sup>17</sup>

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 it 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.<sup>18</sup>

`\ensureascii` `{\text}`

<sup>17</sup>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.

<sup>18</sup>But still defined for backwards compatibility.

**New 3.9i** This macro makes sure  $\langle text \rangle$  is typeset with a LICR-savvy encoding in the ASCII range. It is used to redefine  $\backslash TeX$  and  $\backslash LaTeX$  so that they are correctly typeset even with LGR or X2 (the complete list is stored in  $\backslash BabelNonASCII$ , 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  $\backslash TeX$  and  $\backslash LaTeX$  are not redefined); otherwise,  $\backslash ensureascii$  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 TS3 are not taken into account, since they are not used for “ordinary” text (they are stored in  $\backslash BabelNonText$ , 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 **luatex** 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 engines must wait. This applies to text; there is a basic support for **graphical** elements, including the picture environment (with **pict2e**) and **pfg/tikz**. Also, indexes and the like are under study, as well as math (there are progresses in the latter, including **amsmath** and **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).

**WARNING** If characters to be mirrored are shown without changes with **luatex**, try with the following line:

```
\babeladjust{bidi.mirroring=off}
```

There are some package options controlling **bidi** writing.

**bidi**= default | basic | basic-r | bidi-l | bidi-r

**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 **xetex** and **pdftex** this is the only option.

In **luatex**, **basic-r** provides a simple and fast method for R text, which handles numbers and unmarked L text within an R context many in typical cases. **New 3.19** Finally, **basic** supports both L and R text, and it is the preferred method (support for **basic-r** is currently limited). (They are named **basic** mainly because they only consider the intrinsic direction of scripts and weak directionality.)

**New 3.29** In **xetex**, **bidi-r** and **bidi-l** resort to the package **bidi** (by Vafa Khalighi). Integration is still somewhat tentative, but it mostly works. For RL documents use the former, and for LR ones use the latter.

There are samples on GitHub, under `/required/babel/samples`. See particularly `lua-bidibasic.tex` and `lua-secenum.tex`.

**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 `basic` is available in `luatex` only.

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

    وقد عرفت شبه جزيرة العرب طيلة العصر الهيليني (الاجريقي) بـ
    Arabia أو Aravia (بالاغريقية Αραβία), استخدم الرومان ثلاث
    بادئات بـ "Arabia" على ثلاث مناطق من شبه الجزيرة العربية، إلا أنها
    حقيقةً كانت أكبر مما تعرف عليه اليوم.

\end{document}
```

**EXAMPLE** With `bidi=basic` both L and 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 `bidi=basic-r`, but with R text inside L text you may want to map the font so that the correct features are in force. This is accomplished with an option in `\babelprovide`, as illustrated:

```
\documentclass{book}

\usepackage[english, bidi=basic]{babel}

\babelprovide[onchar=ids fonts]{arabic}

\babelfont{rm}{Crimson}
\babelfont[*arabic]{rm}{FreeSerif}

\begin{document}

    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 فصحى العصر \textit{fuṣḥā l-‘aṣr} (MSA) and
    فصحى التراث \textit{fuṣḥā t-turāth} (CA).

\end{document}
```

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 `\textthe` must be defined to select the main language):

```
\newcommand\refrange[2]{\babelsublr{\textthe{\ref{#1}}-\textthe{\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, `\subsection{<subsection>.<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.<sup>19</sup>

**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:

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

<sup>19</sup>Next on the roadmap are counters and numeral systems in general. Expect some minor readjustments.

`\babelsublr {<lr-text>}`

Digits in pdfTeX must be marked up explicitly (unlike LaTeX 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 `\babelsublr` 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 {<cmd>}{<local-language>}{<before>}{<after>}`

**New 3.17** Something like:

```
\BabelFootnote{\parsfootnote}{\language}\language{({})}
```

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

```
\footnote{(\foreignlanguage{\language}\language){note}}
```

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}{\language}\language{}{}%
\BabelFootnote{\localfootnote}{\language}\language{}{}%
\BabelFootnote{\mainfootnote}{\language}\language{}{}%
```

(which also redefines `\footnotetext` and defines `\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.

**New 3.64** 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 `babel/⟨language-name⟩/⟨event-name⟩` (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.

`\AddBabelHook` [`⟨lang⟩`] {`⟨name⟩`} {`⟨event⟩`} {`⟨code⟩`}

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

**New 3.33** They may be also 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 T<sub>E</sub>X parameters (#1, #2, #3), with the meaning given:

**adddialect** (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⟨language⟩`. This event and the next one should not contain language-dependent code (for that, add it to `\extras⟨language⟩`).

**afterextras** Just after executing `\extras⟨language⟩`. 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⟨language⟩` and `\date⟨language⟩`.

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`.

**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  
**Portuguese** portuguese, brazilian (portuges, brazil)<sup>20</sup>  
**Romanian** romanian  
**Russian** russian  
**Scottish Gaelic** scottish  
**Spanish** spanish  
**Slovakian** slovak  
**Slovenian** slovene  
**Swedish** swedish  
**Serbian** serbian  
**Turkish** turkish  
**Ukrainian** ukrainian  
**Upper Sorbian** uppersorbian  
**Welsh** welsh

There are more languages not listed above, including hindi, thai, thaicjk, latvian, turkmen, magyar, mongolian, romansh, lithuanian, spanglish, vietnamese, japanese, pinyin, arabic, farsi, ibygreek, bgreek, serbianc, frenchle, ethiop and friulan.

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  $\langle file \rangle$ , which creates  $\langle file \rangle$ .tex; you can then typeset the latter with  $\LaTeX$ .

---

<sup>20</sup>The two last name comes from the times when they had to be shortened to 8 characters



## 1.28 Unicode character properties in luatex

**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).

`\babelcharproperty`  $\{\langle char-code \rangle\}[\langle to-char-code \rangle]\{\langle property \rangle\}\{\langle value \rangle\}$

**New 3.32** Here,  $\{\langle char-code \rangle\}$  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:

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

Please, refer to the Unicode standard (Annex #9 and Annex #14) for the meaning of the available codes. For example, en is ‘European number’ and id is ‘ideographic’.

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

```
\babelcharproperty{`,`}{locale}{english}
```

## 1.29 Tweaking some features

`\babeladjust`  $\{\langle key-value-list \rangle\}$

**New 3.36** Sometimes you might need to disable some babel features. Currently this macro understands the following keys (and only for luatex), with values on or off: `bidi.text`, `bidi.mirroring`, `bidi.mapdigits`, `layout.lists`, `layout.tabular`, `linebreak.sea`, `linebreak.cjk`, `justify.arabic`. For example, you can set `\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 `bidi.text`).

## 1.30 Tips, workarounds, known issues and notes

- If you use the document class *book* and you use `\ref` inside the argument of `\chapter` (or just use `\ref` inside `\MakeUppercase`), L<sup>A</sup>T<sub>E</sub>X will keep complaining about an undefined label. To prevent such problems, you can revert to using uppercase labels, you can use `\lowercase{\ref{foo}}` inside the argument of `\chapter`, or, if you will not use shorthands in labels, set the safe option to none or bib.
- Both `ltxdoc` and `babel` use `\AtBeginDocument` to change some catcodes, and `babel` reloads `hline` to make sure `:` has the right one, so if you want to change the catcode of `|` it has to be done using the same method at the proper place, with

```
\AtBeginDocument{\DeleteShortVerb{||}}
```

*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 `hline` (`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:

```
\addto\extrasfrench{\inputencoding{latin1}}
\addto\extrasrussian{\inputencoding{koi8-r}}
```

- For the hyphenation to work correctly, lccodes cannot change, because T<sub>E</sub>X only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.<sup>21</sup> So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of T<sub>E</sub>X, not of babel. Alternatively, you may use `\usesorthands` to activate ' and `\definesorthand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).
- `\bibitem` is out of sync with `\selectlanguage` in the .aux file. The reason is `\bibitem` uses `\immediate` (and others, in fact), while `\selectlanguage` doesn't. There is a similar issue with floats, too. There is no known workaround.
- Babel does not take into account `\normalsfcodes` 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 (ie, with math code "8000) as a shorthand can make T<sub>E</sub>X 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):

**csquotes** Logical markup for quotes.

**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.<sup>22</sup>

But that is the easy part, because they don't require modifying the L<sup>A</sup>T<sub>E</sub>X 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

<sup>21</sup>This explains why L<sup>A</sup>T<sub>E</sub>X assumes the lowercase mapping of T1 and does not provide a tool for multiple mappings. Unfortunately, `\savingshyphcodes` is not a solution either, because lccodes for hyphenation are frozen in the format and cannot be changed.

<sup>22</sup>See for example POSIX, ISO 14652 and the Unicode Common Locale Data Repository (CLDR). Those systems, however, have limited application to T<sub>E</sub>X because their aim is just to display information and not fine typesetting.

“(1)-ból”, but “from (3)” is “(3)-ból”, in Spanish an item labelled “3.<sup>o</sup>” may be referred to as either “ítem 3.<sup>o</sup>” or “3.<sup>er</sup> ítem”, 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 \foreignlanguage\* (a new starred version of \foreignlanguage). For old an deprecated functions, see the babel site.

#### Options for locales loaded on the fly

**New 3.51** \babeladjust{ autoload.options = ... } sets the options when a language is loaded on the fly (by default, no options). A typical value would be import, which defines captions, date, numerals, etc., but ignores the code in the tex file (for example, extended numerals in Greek).

#### Labels

**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

T<sub>E</sub>X and most engines based on it (pdfT<sub>E</sub>X, xetex,  $\epsilon$ -T<sub>E</sub>X, the main exception being luatex) require hyphenation patterns to be preloaded when a format is created (eg, L<sup>A</sup>T<sub>E</sub>X, XeL<sup>A</sup>T<sub>E</sub>X, pdfL<sup>A</sup>T<sub>E</sub>X). babel provides a tool which has become standard in many distributions and based on a “configuration file” named 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).<sup>23</sup> Until 3.9n, this task was delegated to the package luatex-hyphen, by Khaled Hosny, Élie Roux, and Manuel Pégourié-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).<sup>24</sup>

### 2.1 Format

In that file the person who maintains a T<sub>E</sub>X environment has to record for which languages he has hyphenation patterns *and* in which files these are stored<sup>25</sup>. 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 L<sup>A</sup>T<sub>E</sub>X that the hyphenation patterns just processed have to be known under an alternative name. Here is an example:

```
% File      : language.dat
% Purpose   : tell iniTeX what files with patterns to load.
english     english.hyphenations
```

<sup>23</sup>This feature was added to 3.9o, but it was buggy. Both 3.9o and 3.9p are deprecated.

<sup>24</sup>The loader for lua(e)tex 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.

<sup>25</sup>This is because different operating systems sometimes use very different file-naming conventions.

```
=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.<sup>26</sup> For example:

```
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:

```
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 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  $\text{T}_{\text{E}}\text{X}$  users, so the files have to be coded so that they can be read by both  $\text{\LaTeX}$  and plain  $\text{T}_{\text{E}}\text{X}$ . The current format can be checked by looking at the value of the macro `\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 `\<lang>hyphenmins`, `\<lang>captions`, `\<lang>date`, `\<lang>extras` and `\<lang>noextras` (the last two may be left empty); where `<lang>` is either the name of the language definition file or the name of the  $\text{\LaTeX}$  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, `\<lang>date` but not `\<lang>captions` does not raise an error but can lead to unexpected results.
- When a language definition file is loaded, it can define `\l@<lang>` to be a dialect of `\language0` when `\l@<lang>` is undefined.

<sup>26</sup>This is not a new feature, but in former versions it didn't work correctly.

- 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 (eg, 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 /).

Some recommendations:

- The preferred shorthand is ", which is not used in L<sup>A</sup>T<sub>E</sub>X (quotes are entered as `` and ' '). Other good choices are characters which are not used in a certain context (eg, = in an ancient language). Note however =, <, >, : 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 \noextras<lang> except for umlauthigh and friends, \bbl@deactivate, \bbl@(non)frenchspacing, and language-specific macros. Use always, if possible, \bbl@save and \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 \extras<lang>.
- 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 \latintext is deprecated.<sup>27</sup>
- Please, for “private” internal macros do not use the \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 ini templates available on GitHub as a basis. Just make a pull request or 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.

---

<sup>27</sup>But not removed, for backward compatibility.

- 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:

<http://www.texnia.com/incubator.html>. See also

<https://latex3.github.io/babel/guides/list-of-locale-templates.html>.

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-outer version of the macro `\newlanguage`, defined in `plain.tex` version 3.x. Here “language” is used in the T<sub>E</sub>X 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 T<sub>E</sub>X sense of set of hyphenation patterns.

**\<lang>hyphenmins** The macro `\<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.

**\date<lang>** The macro `\date<lang>` defines `\today`.

**\extras<lang>** 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.

**\noextras<lang>** Because we want to let the user switch between languages, but we do not know what state T<sub>E</sub>X might be in after the execution of `\extras<lang>`, a macro that brings T<sub>E</sub>X into a predefined state is needed. It will be no surprise that the name of this macro is `\noextras<lang>`.

**\bbl@declare@ttribute** 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.

**\main@language** To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use `\main@language` 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.

**\ProvidesLanguage** The macro `\ProvidesLanguage` should be used to identify the language definition files. Its syntax is similar to the syntax of the L<sup>A</sup>T<sub>E</sub>X command `\ProvidesPackage`.

**\LdfInit** 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.

**\ldf@quit** 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.

**`\ldf@finish`** 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.

**`\loadlocalcfg`** 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`.

**`\substitutefontfamily`** (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).

```
\ProvidesLanguage{<language>}
    [2016/04/23 v0.0 <Language> support from the babel system]
\LfInit{<language>}{captions<language>}

\ifx\undefined\l@<language>
    \nopatterns{<Language>}
    \adddialect\l@<language>0
\fi

\adddialect\l@<dialect>\l@<language>

\bbld@declare@ttribute{<language>}{<attrib>}{%
    \expandafter\addto\expandafter\extras<language>
    \expandafter{\extras<attrib><language>}%
    \let\captions<language>\captions<attrib><language>}

\providehyphenmins{<language>}{\tw@\thr@@}

\StartBabelCommands*{<language>}{captions}
\SetString\chaptername{<chapter name>}
% More strings

\StartBabelCommands*{<language>}{date}
\SetString\monthinname{<name of first month>}
% More strings

\StartBabelCommands*{<dialect>}{captions}
\SetString\chaptername{<chapter name>}
% More strings

\StartBabelCommands*{<dialect>}{date}
\SetString\monthinname{<name of first month>}
% More strings

\EndBabelCommands

\addto\extras<language>{}
\addto\noextras<language>{}
\let\extras<dialect>\extras<language>
\let\noextras<dialect>\noextras<language>

\ldf@finish{<language>}
```



**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  $\TeX$  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` The command `\bbl@activate` is used to change the way an active character expands.

`\bbl@deactivate` `\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` 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]

`\bbl@remove@special` 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 `\dospecial`.  $\TeX$  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 `⟨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<sup>28</sup>.

`\babel@save` To save the current meaning of any control sequence, the macro `\babel@save` is provided. It takes one argument, `⟨curname⟩`, the control sequence for which the meaning has to be saved.

`\babel@savevariable` A second macro is provided to save the current value of a variable. In this context, anything that is allowed after the `\` the primitive is considered to be a variable. The macro takes one argument, the `⟨variable⟩`.  
The effect of the preceding macros is to append a piece of code to the current definition of `\originalTeX`. When `\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

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

<sup>28</sup>This mechanism was introduced by Bernd Raichle.



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 `etoolbox`, by Philipp Lehman, consider using the tools provided by this package instead of `\addto`.

### 3.7 Macros common to a number of languages

- `\bbl@allowhyphens` In several languages compound words are used. This means that when  $\TeX$  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 `\bbl@allowhyphens` can be used.
- `\allowhyphens` Same as `\bbl@allowhyphens`, but does nothing if the encoding is T1. It is intended mainly for characters provided as real glyphs by this encoding but constructed with `\accent` in OT1.  
Note the previous command (`\bbl@allowhyphens`) has different applications (hyphens and discretionaries) than this one (composite chars). Note also prior to version 3.7, `\allowhyphens` had the behavior of `\bbl@allowhyphens`.
- `\set@low@box` For some languages, quotes need to be lowered to the baseline. For this purpose the macro `\set@low@box` is available. It takes one argument and puts that argument in an `\hbox`, at the baseline. The result is available in `\box0` for further processing.
- `\save@sf@q` Sometimes it is necessary to preserve the `\spacefactor`. For this purpose the macro `\save@sf@q` is available. It takes one argument, saves the current `\spacefactor`, executes the argument, and restores the `\spacefactor`.
- `\bbl@frenchspacing` The commands `\bbl@frenchspacing` and `\bbl@nonfrenchspacing` can be used to properly switch French spacing on and off.
- `\bbl@nonfrenchspacing`

### 3.8 Encoding-dependent strings

**New 3.9a** Babel 3.9 provides a way of defining strings in several encodings, intended mainly for `luatex` and `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 `strings`. If there is no `strings`, these blocks are ignored, except `\SetCases` (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 `\StartBabelCommands`. The last block is closed with `\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`  $\{ \langle \textit{language-list} \rangle \} \{ \langle \textit{category} \rangle \} [ \langle \textit{selector} \rangle ]$

The  $\langle \textit{language-list} \rangle$  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 (ie, 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 an encoded way).

The *category* is either captions, date or extras. You must stick to these three categories, even if no error is raised when using other name.<sup>29</sup> It may be empty, too, but in such a case using \SetString is an error (but not \SetCase).

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

\StartBabelCommands{language}{captions}
\SetString{chaptername}{ascii-maybe-LICR-string}

\EndBabelCommands
```

A real example is:

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

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

\StartBabelCommands{austrian}{date}
\SetString\monthiname{J\"a\"nner}

\StartBabelCommands{german}{date}
\SetString\monthiname{Januar}

\StartBabelCommands{german,austrian}{date}
\SetString\monthiiname{Februar}
\SetString\monthiiname{M\"a\"rz}
\SetString\monthivname{April}
\SetString\monthvname{Mai}
\SetString\monthviname{Juni}
\SetString\monthviiname{Juli}
\SetString\monthviiname{August}
\SetString\monthixname{September}
\SetString\monthxname{Oktober}
\SetString\monthxiname{November}
\SetString\monthxiiname{Dezember}
\SetString\today{\number\day.~%
\csname month\romannumeral\month name\endcsname\space
```

<sup>29</sup>In future releases further categories may be added.

```

\number\year}

\StartBabelCommands{german,austrian}{captions}
\SetString\prefacename{Vorwort}
[etc.]

\EndBabelCommands

```

When used in ldf files, previous values of `\category``<language>` 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 `\date``<language>` exists).

**\StartBabelCommands** `*``{<language-list>}{<category>}[<selector>]`

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.<sup>30</sup>

**\EndBabelCommands** Marks the end of the series of blocks.

**\AfterBabelCommands** `{<code>}`

The code is delayed and executed at the global scope just after `\EndBabelCommands`.

**\SetString** `{<macro-name>}{<string>}`

Adds `<macro-name>` to the current category, and defines globally `<lang-macro-name>` to `<code>` (after applying the transformation corresponding to the current charset or defined with the hook `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.

**\SetStringLoop** `{<macro-name>}{<string-list>}`

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** `[<map-list>]{<toupper-code>}{<tolower-code>}`

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 `\@uclclist` (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  $\TeX$ , we can set for Turkish:

<sup>30</sup>This replaces in 3.9g a short-lived `\UseStrings` which has been removed because it did not work.

```

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

\StartBabelCommands{turkish}{}[unicode, fontenc=TU EU1 EU2, charset=utf8]
\SetCase
  {\uccode`i=`İ\relax
   \uccode`ı=`I\relax}
  {\lccode`İ=`i\relax
   \lccode`I=`ı\relax}

\StartBabelCommands{turkish}{}
\SetCase
  {\uccode`i="9D\relax
   \uccode"19=`I\relax}
  {\lccode"9D=`i\relax
   \lccode`I="19\relax}

\EndBabelCommands

```

(Note the mapping for OT1 is not complete.)

**\SetHyphenMap**  $\langle\text{to-lower-macros}\rangle$

**New 3.9g** Case mapping serves in T<sub>E</sub>X 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 T<sub>E</sub>X primitive (`\lccode`), `babel` sets them separately. There are three helper macros to be used inside `\SetHyphenMap`:

- `\BabelLower $\langle\text{uccode}\rangle\langle\text{lccode}\rangle$`  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 $\langle\text{uccode-from}\rangle\langle\text{uccode-to}\rangle\langle\text{step}\rangle\langle\text{lccode-from}\rangle$`  loops through the given uppercase codes, using the step, and assigns them the `\lccode`, which is also increased (MM stands for *many-to-many*).
- `\BabelLowerMO $\langle\text{uccode-from}\rangle\langle\text{uccode-to}\rangle\langle\text{step}\rangle\langle\text{lccode}\rangle$`  loops through the given uppercase codes, using the step, and assigns them the `\lccode`, which is fixed (MO stands for *many-to-one*).

An example is (which is redundant, because these assignments are done by both `luatex` and `xetex`):

```

\SetHyphenMap{\BabelLowerMM{"100}{ "11F}{2}{ "101}}

```

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

### 3.9 Executing code based on the selector

**\IfBabelSelectorTF**  $\langle\text{selectors}\rangle\langle\text{true}\rangle\langle\text{false}\rangle$

**New 3.67** Sometimes a different setup is desired depending on the selector used. Values allowed in  $\langle\text{selectors}\rangle$  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 `\extras⟨language⟩`.

## Part II

# Source code

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 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](mailto:kadingira@tug.org) on <http://tug.org/mailman/listinfo/kadingira>).

## 4 Identification and loading of required files

*Code documentation is still under revision.*

**The following description is no longer valid, because switch and plain have been merged into babel.def.**

The babel package after unpacking consists of the following files:

**switch.def** defines macros to set and switch languages.

**babel.def** defines the rest of macros. It has two parts: a generic one and a second one only for LaTeX.

**babel.sty** is the  $\TeX$  package, which sets options and loads language styles.

**plain.def** defines some  $\TeX$  macros required by `babel.def` and provides a few tools for Plain.

**hyphen.cfg** is the file to be used when generating the formats to load hyphenation patterns.

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

## 5 locale directory

A required component of 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, 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.

**encodings** 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, MMMM 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

```
1 <<version=3.81.2898>>
2 <<date=2022/10/22>>
```

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

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

```
3 <<*Basic macros>> ≡
4 \bbl@trace{Basic macros}
5 \def\bbl@stripslash{\expandafter\@gobble\string}
6 \def\bbl@add#1#2{%
7   \bbl@ifunset{\bbl@stripslash#1}%
8     {\def#1{#2}}%
9     {\expandafter\def\expandafter#1\expandafter{#1#2}}%
10 \def\bbl@xin@{\@expandtwoargs\in@}
11 \def\bbl@carg#1#2{\expandafter#1\csname#2\endcsname}%
12 \def\bbl@ncarg#1#2#3{\expandafter#1\expandafter#2\csname#3\endcsname}%
13 \def\bbl@ccarg#1#2#3{%
14   \expandafter#1\csname#2\expandafter\endcsname\csname#3\endcsname}%
15 \def\bbl@csarg#1#2{\expandafter#1\csname\bbl@#2\endcsname}%
16 \def\bbl@cs#1{\csname\bbl@#1\endcsname}
17 \def\bbl@c1#1{\csname\bbl@#1\language\endcsname}
18 \def\bbl@loop#1#2#3{\bbl@loop#1{#3}#2,\@nnil,}
19 \def\bbl@loopx#1#2{\expandafter\bbl@loop\expandafter#1\expandafter{#2}}
20 \def\bbl@@loop#1#2#3,{%
21   \ifx\@nnil#3\relax\else
22     \def#1{#3}#2\bbl@afterfi\bbl@@loop#1{#2}%
23   \fi}
24 \def\bbl@for#1#2#3{\bbl@loopx#1{#2}{\ifx#1\@empty\else#3\fi}}
```

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

```
25 \def\bbl@add@list#1#2{%
26   \edef#1{%
27     \bbl@ifunset{\bbl@stripslash#1}%
28     {}%
29     {\ifx#1\@empty\else#1,\fi}%
30     #2}}
```

`\bbl@afterelse` `\bbl@afterfi` Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the `\else` and `\fi` parts of an `\if`-statement<sup>31</sup>. These macros will break if another `\if... \fi` statement appears in one of the arguments and it is not enclosed in braces.

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

`\bbl@exp` Now, just syntactical sugar, but it makes partial expansion of some code a lot more simple and readable. Here `\` stands for `\noexpand`, `\<.>` for `\noexpand` applied to a built macro name (which does not define the macro if undefined to `\relax`, because it is created locally), and `\[...]` for one-level expansion (where `...` is the macro name without the backslash). The result may be followed by extra arguments, if necessary.

```
33 \def\bbl@exp#1{%
```

<sup>31</sup>This code is based on code presented in TUGboat vol. 12, no2, June 1991 in “An expansion Power Lemma” by Sonja Maus.

```

34 \begingroup
35 \let\l\lnoexpand
36 \let\l\lbblexp@en
37 \let\l\lbblexp@ue
38 \edef\bblexp@aux{\endgroup#1}%
39 \bblexp@aux}
40 \def\bblexp@en#1>{\expandafter\lnoexpand\csname#1\endcsname}%
41 \def\bblexp@ue#1]{%
42 \unexpanded\expandafter\expandafter\expandafter{\csname#1\endcsname}}%

```

`\bblexp@trim` The following piece of code is stolen (with some changes) from `keyval`, by David Carlisle. It defines two macros: `\bblexp@trim` and `\bblexp@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.

```

43 \def\bblexp@tempa#1{%
44 \long\def\bblexp@trim##1##2{%
45 \futurelet\bblexp@trim@a\bblexp@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
46 \def\bblexp@trim@c{%
47 \ifx\bblexp@trim@a\@sptoken
48 \expandafter\bblexp@trim@b
49 \else
50 \expandafter\bblexp@trim@b\expandafter#1%
51 \fi}%
52 \long\def\bblexp@trim@b#1##1 \@nil{\bblexp@trim@i##1}}
53 \bblexp@tempa{ }
54 \long\def\bblexp@trim@i#1\@nil#2\relax#3{#3{#1}}
55 \long\def\bblexp@trim@def#1{\bblexp@trim{\def#1}}

```

`\bblexp@ifunset` To check if a macro is defined, we create a new macro, which does the same as `\@ifundefined`. However, in an `ε`-tex engine, it is based on `\ifcsname`, which is more efficient, and does not waste memory. Defined inside a group, to avoid `\ifcsname` being implicitly set to `\relax` by the `\csname` test.

```

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

```

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

```

76 \def\bblexp@ifblank#1{%
77 \bblexp@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
78 \long\def\bblexp@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
79 \def\bblexp@ifset#1#2#3{%
80 \bblexp@ifunset{#1}{#3}{\bblexp{\l\lbblexp@ifblank{\@nameuse{#1}}}{#3}{#2}}}%

```

For each element in the comma separated `<key>=<value>` list, execute `<code>` with #1 and #2 as the key and the value of current item (trimmed). In addition, the item is passed verbatim as #3. With the

<key> alone, it passes \@empty (ie, the macro thus named, not an empty argument, which is what you get with <key>= and no value).

```

81 \def\bbl@forkv#1#2{%
82   \def\bbl@kvcmd##1##2##3{#2}%
83   \bbl@kvnext#1,\@nil,}
84 \def\bbl@kvnext#1,{%
85   \ifx\@nil#1\relax\else
86     \bbl@ifblank{#1}{\bbl@forkv@eq#1=\@empty=\@nil{#1}}%
87     \expandafter\bbl@kvnext
88   \fi}
89 \def\bbl@forkv@eq#1=#2=#3\@nil#4{%
90   \bbl@trim\def\bbl@forkv@a{#1}%
91   \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).

```

92 \def\bbl@vforeach#1#2{%
93   \def\bbl@forcmd##1{#2}%
94   \bbl@fornext#1,\@nil,}
95 \def\bbl@fornext#1,{%
96   \ifx\@nil#1\relax\else
97     \bbl@ifblank{#1}{\bbl@trim\bbl@forcmd{#1}}%
98     \expandafter\bbl@fornext
99   \fi}
100 \def\bbl@foreach#1{\expandafter\bbl@vforeach\expandafter{#1}}

```

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

```

101 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
102   \toks@{}%
103   \def\bbl@replace@aux##1#2##2#2{%
104     \ifx\bbl@nil##2%
105       \toks@\expandafter{\the\toks@##1}%
106     \else
107       \toks@\expandafter{\the\toks@##1#3}%
108       \bbl@afterfi
109       \bbl@replace@aux##2#2%
110     \fi}%
111   \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
112   \edef#1{\the\toks@}}

```

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

```

113 \ifx\detokenize\undefined\else % Unused macros if old Plain TeX
114   \bbl@exp{\def\\bbl@parsedef##1\detokenize{macro:}}#2->#3\relax{%
115     \def\bbl@tempa{#1}%
116     \def\bbl@tempb{#2}%
117     \def\bbl@tempe{#3}}
118   \def\bbl@sreplace#1#2#3{%
119     \begingroup
120     \expandafter\bbl@parsedef\meaning#1\relax
121     \def\bbl@tempc{#2}%
122     \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
123     \def\bbl@tempd{#3}%
124     \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
125     \bbl@xin@{\bbl@tempc}{\bbl@tempe}% If not in macro, do nothing
126     \ifin@
127       \bbl@exp{\\bbl@replace\\bbl@tempe{\bbl@tempc}{\bbl@tempd}}%
128       \def\bbl@tempc{% Expanded an executed below as 'uplevel'
129         \\makeatletter % "internal" macros with @ are assumed
130         \\scantokens{%

```



```

131      \bbl@tempa\\@namedef{\bbl@stripslash#1}\bbl@tempb{\bbl@tempe}}%
132      \catcode64=\the\catcode64\relax}% Restore @
133      \else
134      \let\bbl@tempc\@empty % Not \relax
135      \fi
136      \bbl@exp{% For the 'uplevel' assignments
137      \endgroup
138      \bbl@tempc}} % empty or expand to set #1 with changes
139 \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 pdf<sub>ε</sub>X, 1 is luatex, and 2 is xetex. You may use the latter it in your language style if you want.

```

140 \def\bbl@ifsamestring#1#2{%
141   \begingroup
142   \protected@edef\bbl@tempb{#1}%
143   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
144   \protected@edef\bbl@tempc{#2}%
145   \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
146   \ifx\bbl@tempb\bbl@tempc
147     \aftergroup\@firstoftwo
148   \else
149     \aftergroup\@secondoftwo
150   \fi
151 \endgroup}
152 \chardef\bbl@engine=%
153 \ifx\directlua\@undefined
154   \ifx\XeTeXinputencoding\@undefined
155     \z@
156   \else
157     \tw@
158   \fi
159 \else
160   \@ne
161 \fi

```

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

```

162 \def\bbl@bsphack{%
163   \ifhmode
164     \hskip\z@skip
165     \def\bbl@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
166   \else
167     \let\bbl@esphack\@empty
168   \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`.

```

169 \def\bbl@cased{%
170   \ifx\oe\OE
171     \expandafter\in@\expandafter
172     {\expandafter\OE\expandafter}\expandafter{\oe}%
173     \ifin@
174       \bbl@afterelse\expandafter\MakeUppercase
175     \else
176       \bbl@afterfi\expandafter\MakeLowercase
177     \fi
178   \else
179     \expandafter\@firstofone
180   \fi}

```

An alternative to `\IfFormatAtLeastTF` for old versions. Temporary.

```

181 \ifx\IfFormatAtLeastTF\@undefined
182   \def\bbl@ifformatlater{\@ifl@t@r\fmtversion}

```

```

183 \else
184   \let\bbl@ifformatlater\IfFormatAtLeastTF
185 \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`).

```

186 \def\bbl@extras@wrap#1#2#3{% 1:in-test, 2:before, 3:after
187   \toks@\expandafter\expandafter\expandafter{%
188     \csname extras\language\endcsname}%
189   \bbl@exp{\in@{#1}{\the\toks@}}%
190   \ifin@ \else
191     \@temptokena{#2}%
192     \edef\bbl@tempc{\the\@temptokena\the\toks@}%
193     \toks@\expandafter\bbl@tempc#3}%
194     \expandafter\edef\csname extras\language\endcsname{\the\toks@}%
195   \fi}
196 <\/Basic macros>

```

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

```

197 <\/*Make sure ProvidesFile is defined> \equiv
198 \ifx\ProvidesFile\@undefined
199   \def\ProvidesFile#1[#2 #3 #4]{%
200     \wlog{File: #1 #4 #3 <#2>}%
201     \let\ProvidesFile\@undefined}
202 \fi
203 <\/Make sure ProvidesFile is defined>

```

## 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 require loading `switch.def` in the format.

```

204 <\/*Define core switching macros> \equiv
205 \ifx\language\@undefined
206   \csname newcount\endcsname\language
207 \fi
208 <\/Define core switching macros>

```

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

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

```

209 <\/*Define core switching macros> \equiv
210 \countdef\last@language=19
211 \def\addlanguage{\csname newlanguage\endcsname}
212 <\/Define core switching macros>

```

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

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

## 6.2 The Package File (LaTeX, babel.sty)

```

213 <package>
214 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
215 \ProvidesPackage{babel}[\<date>] \<version>] The Babel package]

```

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

```

216 \@ifpackagewith{babel}{debug}
217   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}}%
218   \let\bbl@debug\@firstofone
219   \ifx\directlua\@undefined\else
220     \directlua{ Babel = Babel or {}
221       Babel.debug = true }%
222     \input{babel-debug.tex}%
223   \fi}
224 {\providecommand\bbl@trace[1]}%
225 \let\bbl@debug\@gobble
226 \ifx\directlua\@undefined\else
227   \directlua{ Babel = Babel or {}
228     Babel.debug = false }%
229   \fi}
230 \def\bbl@error#1#2{%
231   \begingroup
232     \def\{\MessageBreak}%
233     \PackageError{babel}{#1}{#2}%
234   \endgroup}
235 \def\bbl@warning#1{%
236   \begingroup
237     \def\{\MessageBreak}%
238     \PackageWarning{babel}{#1}%
239   \endgroup}
240 \def\bbl@infowarn#1{%
241   \begingroup
242     \def\{\MessageBreak}%
243     \PackageNote{babel}{#1}%
244   \endgroup}
245 \def\bbl@info#1{%
246   \begingroup
247     \def\{\MessageBreak}%
248     \PackageInfo{babel}{#1}%
249   \endgroup}

```

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

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

```

250 <Basic macros>
251 \@ifpackagewith{babel}{silent}
252   {\let\bbl@info\@gobble
253     \let\bbl@infowarn\@gobble
254     \let\bbl@warning\@gobble}
255   {}
256 %
257 \def\AfterBabelLanguage#1{%
258   \global\expandafter\bbl@add\csname#1.ldf-h@@k\endcsname}%

```

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

```

259 \ifx\bbl@languages\@undefined\else
260   \begingroup
261     \catcode\^^I=12
262     \@ifpackagewith{babel}{showlanguages}{%
263       \begingroup

```

```

264         \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}}%
265         \wlog{<*languages>}%
266         \bbl@languages
267         \wlog{</languages>}%
268     \endgroup{}}
269 \endgroup
270 \def\bbl@elt#1#2#3#4{%
271     \ifnum#2=\z@
272         \gdef\bbl@nulllanguage{#1}%
273         \def\bbl@elt##1##2##3##4{%
274             \fi}%
275     \bbl@languages
276 \fi%

```

### 6.3 base

The first ‘real’ option to be processed is base, which set the hyphenation patterns then resets `ver@babel.sty` so that  $\TeX$  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`.

```

277 \bbl@trace{Defining option 'base'}
278 \@ifpackagewith{babel}{base}{%
279     \let\bbl@onlyswitch\@empty
280     \let\bbl@provide@locale\relax
281     \input babel.def
282     \let\bbl@onlyswitch\@undefined
283     \ifx\directlua\@undefined
284         \DeclareOption*{\bbl@patterns{\CurrentOption}}%
285     \else
286         \input luababel.def
287         \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
288     \fi
289     \DeclareOption{base}{}%
290     \DeclareOption{showlanguages}{}%
291     \ProcessOptions
292     \global\expandafter\let\csname opt@babel.sty\endcsname\relax
293     \global\expandafter\let\csname ver@babel.sty\endcsname\relax
294     \global\let@ifl@ter@@\@ifl@ter
295     \def@ifl@ter#1#2#3#4#5{\global\let@ifl@ter\@ifl@ter@@}%
296     \endinput{}%

```

### 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.

```

297 \bbl@trace{key=value and another general options}
298 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
299 \def\bbl@tempb#1.#2{% Remove trailing dot
300     #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
301 \def\bbl@tempd#1.#2\@nnil{% TODO. Refactor lists?
302     \ifx\@empty#2%
303         \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
304     \else
305         \in@{,provide=}{, #1}%
306         \ifin@
307             \edef\bbl@tempc{%
308                 \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
309         \else
310             \in@{=}{#1}%
311             \ifin@

```

```

312 \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
313 \else
314 \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
315 \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
316 \fi
317 \fi
318 \fi}
319 \let\bbl@tempc\@empty
320 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
321 \expandafter\let\csname opt@babel.sty\endcsname\bbl@tempc

```

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

```

322 \DeclareOption{KeepShorthandsActive}{}
323 \DeclareOption{activeacute}{}
324 \DeclareOption{activegrave}{}
325 \DeclareOption{debug}{}
326 \DeclareOption{noconfigs}{}
327 \DeclareOption{showlanguages}{}
328 \DeclareOption{silent}{}
329 % \DeclareOption{mono}{}
330 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
331 \chardef\bbl@iniflag\z@
332 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main -> +1
333 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\@tw@} % add = 2
334 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\@thr@} % add + main
335 % A separate option
336 \let\bbl@autoload@options\@empty
337 \DeclareOption{provide@=*}{\def\bbl@autoload@options{import}}
338 % Don't use. Experimental. TODO.
339 \newif\ifbbl@single
340 \DeclareOption{selectors=off}{\bbl@singletrue}
341 <(More package options)>

```

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

```

342 \let\bbl@opt@shorthands\@nnil
343 \let\bbl@opt@config\@nnil
344 \let\bbl@opt@main\@nnil
345 \let\bbl@opt@headfoot\@nnil
346 \let\bbl@opt@layout\@nnil
347 \let\bbl@opt@provide\@nnil

```

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

```

348 \def\bbl@tempa#1=#2\bbl@tempa{%
349 \bbl@csarg\ifx{opt@#1}\@nnil
350 \bbl@csarg\edef{opt@#1}{#2}%
351 \else
352 \bbl@error
353 {Bad option '#1=#2'. Either you have misspelled the\\%
354 key or there is a previous setting of '#1'. Valid\\%
355 keys are, among others, 'shorthands', 'main', 'bidi',\\%
356 'strings', 'config', 'headfoot', 'safe', 'math'.}%
357 {See the manual for further details.}
358 \fi}

```

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

```

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

```

```

361 \bbl@xin@{\string=}{\CurrentOption}%
362 \ifin@
363 \expandafter\bbl@tempa\CurrentOption\bbl@tempa
364 \else
365 \bbl@add@list\bbl@language@opts{\CurrentOption}%
366 \fi}

```

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

```

367 \ProcessOptions*
368 \ifx\bbl@opt@provide\@nnil
369 \let\bbl@opt@provide\@empty %%% MOVE above
370 \else
371 \chardef\bbl@iniflag\@ne
372 \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
373 \in@{,provide,}{, #1,}%
374 \ifin@
375 \def\bbl@opt@provide{#2}%
376 \bbl@replace\bbl@opt@provide{;}{,}%
377 \fi}
378 \fi
379 %

```

## 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=...

```

380 \bbl@trace{Conditional loading of shorthands}
381 \def\bbl@sh@string#1{%
382 \ifx#1\@empty\else
383 \ifx#1t\string~%
384 \else\ifx#1c\string,%
385 \else\string#1%
386 \fi\fi
387 \expandafter\bbl@sh@string
388 \fi}
389 \ifx\bbl@opt@shorthands\@nnil
390 \def\bbl@ifshorthand#1#2#3{#2}%
391 \else\ifx\bbl@opt@shorthands\@empty
392 \def\bbl@ifshorthand#1#2#3{#3}%
393 \else

```

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

```

394 \def\bbl@ifshorthand#1{%
395 \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
396 \ifin@
397 \expandafter\@firstoftwo
398 \else
399 \expandafter\@secondoftwo
400 \fi}

```

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

```

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

```

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

```

403 \bbl@ifshorthand{'}%
404 {\PassOptionsToPackage{activeacute}{babel}}{}
405 \bbl@ifshorthand{`}%
406 {\PassOptionsToPackage{activegrave}{babel}}{}
407 \fi\fi

```

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.

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

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

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

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

```
418 \bbl@trace{Defining IfBabelLayout}
419 \ifx\bbl@opt@layout\@nnil
420   \newcommand\IfBabelLayout[3]{#3}%
421 \else
422   \newcommand\IfBabelLayout[1]{%
423     \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
424     \ifin@
425       \expandafter\@firstoftwo
426     \else
427       \expandafter\@secondoftwo
428     \fi}
429 \fi
430 \</package>
431 \<core>
```

## 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 *Emulate LaTeX*.

```
432 \ifx\ldf@quit\@undefined\else
433 \endinput\fi % Same line!
434 \<Make sure ProvidesFile is defined>
435 \ProvidesFile{babel.def}[\<date>] [\<version>] Babel common definitions]
436 \ifx\AtBeginDocument\@undefined % TODO. change test.
437   \<Emulate LaTeX>
438 \fi
```

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

```
439 \</core>
440 \<package | core>
```

## 7 Multiple languages

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

Plain  $\text{\TeX}$  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.

```
441 \def\bbl@version{\<version>}
442 \def\bbl@date{\<date>}
443 \<Define core switching macros>
```

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

```

444 \def\adddialect#1#2{%
445   \global\chardef#1#2\relax
446   \bbl@usehooks{\adddialect}{#1}{#2}}%
447   \begingroup
448     \count@#1\relax
449     \def\bbl@elt##1##2##3##4{%
450       \ifnum\count@=##2\relax
451         \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
452         \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
453           set to \expandafter\string\csname l@##1\endcsname\%
454             (\string\language\the\count@). Reported}%
455         \def\bbl@elt####1####2####3####4{%
456           \fi}%
457         \bbl@cs{languages}%
458       \endgroup

```

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

```

459 \def\bbl@fixname#1{%
460   \begingroup
461   \def\bbl@tempe{l@}%
462   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
463   \bbl@tempd
464     {\lowercase\expandafter{\bbl@tempd}%
465     {\uppercase\expandafter{\bbl@tempd}%
466     \@empty
467     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
468     {\uppercase\expandafter{\bbl@tempd}}}%
469     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
470     {\lowercase\expandafter{\bbl@tempd}}}%
471     \@empty
472     \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
473   \bbl@tempd
474   \bbl@exp{\bbl@usehooks{language}{\language}{#1}}%
475 \def\bbl@iflanguage#1{%
476   \@ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}

```

After a name has been ‘fixed’, the selectors will try to load the language. If even the fixed name is not defined, will load it on the fly, either based on its name, or if activated, its BCP47 code.

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

```

477 \def\bbl@bcpcase#1#2#3#4\@#5{%
478   \ifx\@empty#3%
479     \uppercase{\def#5{#1#2}}%
480   \else
481     \uppercase{\def#5{#1}}%
482     \lowercase{\edef#5{#5#2#3#4}}%
483   \fi}
484 \def\bbl@bcpllookup#1-#2-#3-#4\@#5{%
485   \let\bbl@bcp\relax
486   \lowercase{\def\bbl@tempa{#1}}%
487   \ifx\@empty#2%
488     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
489   \else\ifx\@empty#3%
490     \bbl@bcpcase#2\@empty\@empty\@#5\bbl@tempb
491     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%

```





```

550 \def\iflanguage#1{%
551   \bbl@iflanguage{#1}{%
552     \ifnum\csname l@#1\endcsname=\language
553       \expandafter\@firstoftwo
554     \else
555       \expandafter\@secondoftwo
556     \fi}}

```

## 7.1 Selecting the language

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

```

557 \let\bbl@select@type\z@
558 \edef\selectlanguage{%
559   \noexpand\protect
560   \expandafter\noexpand\csname selectlanguage \endcsname}

```

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

```

561 \ifx\@undefined\protect\let\protect\relax\fi

```

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

```

562 \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.

```

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

```

564 \def\bbl@push@language{%
565   \ifx\language\@undefined\else
566     \ifx\currentgrouplevel\@undefined
567       \xdef\bbl@language@stack{\language+\bbl@language@stack}%
568     \else
569       \ifnum\currentgrouplevel=\z@
570         \xdef\bbl@language@stack{\language+}%
571       \else
572         \xdef\bbl@language@stack{\language+\bbl@language@stack}%
573       \fi
574     \fi
575   \fi}

```

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

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

```

576 \def\bbl@pop@lang#1+#2\@@{%
577   \edef\language{#1}%
578   \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).

```
579 \let\bbl@ifrestoring\@secondoftwo
580 \def\bbl@pop@language{%
581   \expandafter\bbl@pop@lang\bbl@language@stack\@@
582   \let\bbl@ifrestoring\@firstoftwo
583   \expandafter\bbl@set@language\expandafter{\language}%
584   \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).

```
585 \chardef\localeid\z@
586 \def\bbl@id@last{0} % No real need for a new counter
587 \def\bbl@id@assign{%
588   \bbl@ifunset\bbl@id@\language%
589   {\count@\bbl@id@last\relax
590    \advance\count@\@ne
591    \bbl@csarg\chardef{id@\language}\count@
592    \edef\bbl@id@last{\the\count@}%
593    \ifcase\bbl@engine\or
594      \directlua{
595        Babel = Babel or {}
596        Babel.locale_props = Babel.locale_props or {}
597        Babel.locale_props[\bbl@id@last] = {}
598        Babel.locale_props[\bbl@id@last].name = '\language'
599      }%
600    \fi}%
601  }%
602  \chardef\localeid\bbl@cl{id@}}
```

The unprotected part of `\selectlanguage`.

```
603 \expandafter\def\csname selectlanguage \endcsname#1{%
604   \ifnum\bbl@hymapsel=\ccclv\let\bbl@hymapsel\tw@\fi
605   \bbl@push@language
606   \aftergroup\bbl@pop@language
607   \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 of `\language`. To catch either form a trick is used, but unfortunately as a side effect the catcodes of letters in `\language` are messed up. This is a bug, but preserved for backwards compatibility. The list of auxiliary files can be extended by redefining `\BabelContentsFiles`, but make sure they are loaded inside a group (as `aux`, `toc`, `lof`, and `lot` do) or the last language of the document will remain active afterwards.

We also write a command to change the current language in the auxiliary files.

`\bbl@savelastskip` is used to deal with skips before the write whatsit (as suggested by U Fischer). Adapted from `hyperref`, but it might fail, so I'll consider it a temporary hack, while I study other options (the ideal, but very likely unfeasible except perhaps in `lualatex`, is to avoid the `\write` altogether when not needed).

```
608 \def\BabelContentsFiles{toc,lof,lot}
609 \def\bbl@set@language#1{% from selectlanguage, pop@
610   % The old buggy way. Preserved for compatibility.
611   \edef\language{%
612     \ifnum\escapechar=\expandafter\string#1\@empty
613     \else\string#1\@empty\fi}%

```

```

614 \ifcat\relax\noexpand#1%
615 \expandafter\ifx\csname date\language\endcsname\relax
616 \edef\language{#1}%
617 \let\locale\language
618 \else
619 \bbl@info{Using '\string\language' instead of 'language' is\\%
620 deprecated. If what you want is to use a\\%
621 macro containing the actual locale, make\\%
622 sure it does not not match any language.\\%
623 Reported}%
624 \ifx\scantokens\undefined
625 \def\locale{??}%
626 \else
627 \scantokens\expandafter\expandafter
628 \def\expandafter\locale\expandafter{\language}%
629 \fi
630 \fi
631 \else
632 \def\locale{#1}% This one has the correct catcodes
633 \fi
634 \select@language{\language}%
635 % write to aux
636 \expandafter\ifx\csname date\language\endcsname\relax\else
637 \if@filesw
638 \ifx\babel@aux@gobbletwo\else % Set if single in the first, redundant
639 \bbl@savelastskip
640 \protected@write\auxout{}\string\babel@aux{\bbl@auxname}{}}%
641 \bbl@restorelastskip
642 \fi
643 \bbl@usehooks{write}{}%
644 \fi
645 \fi}
646 %
647 \let\bbl@restorelastskip\relax
648 \let\bbl@savelastskip\relax
649 %
650 \newif\ifbbl@bcpallowed
651 \bbl@bcpallowedfalse
652 \def\select@language#1{% from set@, babel@aux
653 \ifx\bbl@select@name\empty
654 \def\bbl@select@name{select}%
655 % set hmap
656 \fi
657 \ifnum\bbl@hmapsel=\@cclv\chardef\bbl@hmapsel4\relax\fi
658 % set name
659 \edef\language{#1}%
660 \bbl@fixname\language
661 % TODO. name@map must be here?
662 \bbl@provide@locale
663 \bbl@iflanguage\language{%
664 \let\bbl@select@type\z@
665 \expandafter\bbl@switch\expandafter{\language}}%
666 \def\babel@aux#1#2{%
667 \select@language{#1}%
668 \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
669 \writefile{##1}{\babel@toc{#1}{#2}\relax}}}% TODO - plain?
670 \def\babel@toc#1#2{%
671 \select@language{#1}}

```

First, check if the user asks for a known language. If so, update the value of `\language` and call `\originalTeX` to bring `TEX` in a certain pre-defined state. The name of the language is stored in the control sequence `\language`. Then we have to *redefine* `\originalTeX` to compensate for the things that have been activated. To

save memory space for the macro definition of `\originalTeX`, we construct the control sequence name for the `\noextras<lang>` command at definition time by expanding the `\csname` primitive. Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of `\selectlanguage`, and calling these macros.

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

```

672 \newif\ifbbl@usedatagroup
673 \def\bbl@switch#1{% from select@, foreign@
674   % make sure there is info for the language if so requested
675   \bbl@ensureinfo{#1}%
676   % restore
677   \originalTeX
678   \expandafter\def\expandafter\originalTeX\expandafter{%
679     \csname noextras#1\endcsname
680     \let\originalTeX\@empty
681     \babel@beginsave}%
682   \bbl@usehooks{afterreset}}}%
683   \languageshorthands{none}%
684   % set the locale id
685   \bbl@id@assign
686   % switch captions, date
687   % No text is supposed to be added here, so we remove any
688   % spurious spaces.
689   \bbl@bsphack
690   \ifcase\bbl@select@type
691     \csname captions#1\endcsname\relax
692     \csname date#1\endcsname\relax
693   \else
694     \bbl@xin@{,captions,},{, \bbl@select@opts,}%
695     \ifin@
696       \csname captions#1\endcsname\relax
697     \fi
698     \bbl@xin@{,date,},{, \bbl@select@opts,}%
699     \ifin@ % if \foreign... within \<lang>date
700       \csname date#1\endcsname\relax
701     \fi
702   \fi
703   \bbl@esphack
704   % switch extras
705   \bbl@usehooks{beforeextras}}}%
706   \csname extras#1\endcsname\relax
707   \bbl@usehooks{afterextras}}}%
708   % > babel-ensure
709   % > babel-sh-<short>
710   % > babel-bidi
711   % > babel-fontspec
712   % hyphenation - case mapping
713   \ifcase\bbl@opt@hyphenmap\or
714     \def\BabelLower##1##2{\lccode##1=##2\relax}%
715     \ifnum\bbl@hymapsel>4\else
716       \csname\language @bbl@hyphenmap\endcsname
717       \fi
718     \chardef\bbl@opt@hyphenmap\z@
719   \else
720     \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
721       \csname\language @bbl@hyphenmap\endcsname
722       \fi
723     \fi
724     \let\bbl@hymapsel\@cclv
725     % hyphenation - select rules
726     \ifnum\csname l@ \language\endcsname=\l@unhyphenated

```

```

727 \edef\bbl@tempa{u}%
728 \else
729 \edef\bbl@tempa{\bbl@cl{lbrk}}%
730 \fi
731 % linebreaking - handle u, e, k (v in the future)
732 \bbl@xin@{/u}{/\bbl@tempa}%
733 \ifin@ \else \bbl@xin@{/e}{/\bbl@tempa} \fi % elongated forms
734 \ifin@ \else \bbl@xin@{/k}{/\bbl@tempa} \fi % only kashida
735 \ifin@ \else \bbl@xin@{/p}{/\bbl@tempa} \fi % padding (eg, Tibetan)
736 \ifin@ \else \bbl@xin@{/v}{/\bbl@tempa} \fi % variable font
737 \ifin@
738 % unhyphenated/kashida/elongated/padding = allow stretching
739 \language\l@unhyphenated
740 \babel@savevariable\emergencystretch
741 \emergencystretch\maxdimen
742 \babel@savevariable\hbadness
743 \hbadness\@M
744 \else
745 % other = select patterns
746 \bbl@patterns{#1}%
747 \fi
748 % hyphenation - mins
749 \babel@savevariable\lefthyphenmin
750 \babel@savevariable\righthyphenmin
751 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
752 \set@hyphenmins\tw@\thr@@\relax
753 \else
754 \expandafter\expandafter\expandafter\set@hyphenmins
755 \csname #1hyphenmins\endcsname\relax
756 \fi
757 \let\bbl@selectorname\@empty}

```

`otherlanguage (env.)` The other language 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.

```

758 \long\def\otherlanguage#1{%
759 \def\bbl@selectorname{other}%
760 \ifnum\bbl@hymapsel=\@cc1\let\bbl@hymapsel\thr@@\fi
761 \csname selectlanguage\endcsname{#1}%
762 \ignorespaces}

```

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

```

763 \long\def\endotherlanguage{%
764 \global\@ignoretrue\ignorespaces}

```

`otherlanguage* (env.)` The other language 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`.

```

765 \expandafter\def\csname otherlanguage*\endcsname{%
766 \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
767 \def\bbl@otherlanguage@s[#1]#2{%
768 \def\bbl@selectorname{other*}%
769 \ifnum\bbl@hymapsel=\@cc1\chardef\bbl@hymapsel4\relax\fi
770 \def\bbl@select@opts{#1}%
771 \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”.

```

772 \expandafter\let\csname endotherlanguage*\endcsname\relax

```

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

Unlike `\selectlanguage` this command doesn't switch *everything*, it only switches the hyphenation rules and the extra definitions for the language specified. It does this within a group and assumes the `\extras{lang}` command doesn't make any `\global` changes. The coding is very similar to part of `\selectlanguage`.

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

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

(3.11) Also experimental are the hook `foreign` and `foreign*`. With them you can redefine `\BabelText` which by default does nothing. Its behavior is not well defined yet. So, use it in horizontal mode only if you do not want surprises.

In other words, at the beginning of a paragraph `\foreignlanguage` enters into `hmode` with the surrounding lang, and with `\foreignlanguage*` with the new lang.

```

773 \providecommand\bbl@beforeforeign{}
774 \edef\foreignlanguage{%
775   \noexpand\protect
776   \expandafter\noexpand\csname foreignlanguage \endcsname}
777 \expandafter\def\csname foreignlanguage \endcsname{%
778   \@ifstar\bbl@foreign@s\bbl@foreign@x}
779 \providecommand\bbl@foreign@x[3][]{%
780   \begingroup
781     \def\bbl@selectorname{foreign}%
782     \def\bbl@select@opts{#1}%
783     \let\BabelText\@firstofone
784     \bbl@beforeforeign
785     \foreign@language{#2}%
786     \bbl@usehooks{foreign}{}%
787     \BabelText{#3}% Now in horizontal mode!
788   \endgroup}
789 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
790   \begingroup
791     {\par}%
792     \def\bbl@selectorname{foreign*}%
793     \let\bbl@select@opts\@empty
794     \let\BabelText\@firstofone
795     \foreign@language{#1}%
796     \bbl@usehooks{foreign*}{}%
797     \bbl@dirparastext
798     \BabelText{#2}% Still in vertical mode!
799   {\par}%
800   \endgroup}

```

`\foreign@language` 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`.

```

801 \def\foreign@language#1{%
802   % set name
803   \edef\language#1}%
804 \ifbbl@usedategroup
805   \bbl@add\bbl@select@opts{,date,}%
806   \bbl@usedategroupfalse
807 \fi
808 \bbl@fixname\language
809 % TODO. name@map here?
810 \bbl@provide@locale

```

```

811 \bbl@iflanguage\language\language\name{%
812 \let\bbl@select@type\@ne
813 \expandafter\bbl@switch\expandafter{\language\name}}

```

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

```

814 \def\IfBabelSelectorTF#1{%
815 \bbl@xin@{\bbl@selectorname,}{,\zap@space#1 \@empty,}%
816 \ifin@
817 \expandafter\@firstoftwo
818 \else
819 \expandafter\@secondoftwo
820 \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 `\lcode's` has been set, too). `\bbl@hyphenation@` is set to relax until the very first `\babelhyphenation`, so do nothing with this value. If the exceptions for a language (by its number, not its name, so that `:ENC` is taken into account) has been set, then use `\hyphenation` with both global and language exceptions and empty the latter to mark they must not be set again.

```

821 \let\bbl@hyphlist\@empty
822 \let\bbl@hyphenation@\relax
823 \let\bbl@pttnlist\@empty
824 \let\bbl@patterns@\relax
825 \let\bbl@hymapsel=\cclv
826 \def\bbl@patterns#1{%
827 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
828 \csname l@#1\endcsname
829 \edef\bbl@tempa{#1}%
830 \else
831 \csname l@#1:\f@encoding\endcsname
832 \edef\bbl@tempa{#1:\f@encoding}%
833 \fi
834 \@expandtwoargs\bbl@usehooks{patterns}{\{#1\}\bbl@tempa}}%
835 % > luatex
836 \@ifundefined{bbl@hyphenation@}{\% Can be \relax!
837 \begingroup
838 \bbl@xin@{\number\language,}{,\bbl@hyphlist}%
839 \ifin@\else
840 \@expandtwoargs\bbl@usehooks{hyphenation}{\{#1\}\bbl@tempa}}%
841 \hyphenation{%
842 \bbl@hyphenation@
843 \@ifundefined{bbl@hyphenation@#1}%
844 \@empty
845 {\space\csname bbl@hyphenation@#1\endcsname}}%
846 \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
847 \fi
848 \endgroup}}

```

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

```

849 \def\hyphenrules#1{%
850 \edef\bbl@tempf{#1}%
851 \bbl@fixname\bbl@tempf
852 \bbl@iflanguage\bbl@tempf{%
853 \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
854 \ifx\languageshortands\@undefined\else
855 \languageshortands{none}%
856 \fi
857 \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax

```



```

858 \set@hyphenmins\tw@\thr@\relax
859 \else
860 \expandafter\expandafter\expandafter\set@hyphenmins
861 \csname\bbl@tempf hyphenmins\endcsname\relax
862 \fi}}
863 \let\endhyphenrules\empty

```

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

```

864 \def\providehyphenmins#1#2{%
865 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
866 \@namedef{#1hyphenmins}{#2}%
867 \fi}

```

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

```

868 \def\set@hyphenmins#1#2{%
869 \lefthyphenmin#1\relax
870 \righthyphenmin#2\relax}

```

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

```

871 \ifx\ProvidesFile\undefined
872 \def\ProvidesLanguage#1[#2 #3 #4]{%
873 \wlog{Language: #1 #4 #3 <#2>}%
874 }
875 \else
876 \def\ProvidesLanguage#1{%
877 \begingroup
878 \catcode`\ 10 %
879 \@makeother\/%
880 \@ifnextchar[%]
881 {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
882 \def\@provideslanguage#1[#2]{%
883 \wlog{Language: #1 #2}%
884 \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
885 \endgroup}
886 \fi

```

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

```

887 \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.

```

888 \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’:

```

889 \providecommand\setlocale{%
890 \bbl@error
891 {Not yet available}%
892 {Find an armchair, sit down and wait}}
893 \let\uselocale\setlocale
894 \let\locale\setlocale
895 \let\selectlocale\setlocale
896 \let\textlocale\setlocale
897 \let\textlanguage\setlocale
898 \let\languagetext\setlocale

```

## 7.2 Errors

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

`\@noopterr` When the package was loaded without options not everything will work as expected. An error message is issued in that case.  
When the format knows about `\PackageError` it must be  $\text{\LaTeX 2}_{\epsilon}$ , so we can safely use its error handling interface. Otherwise we'll have to 'keep it simple'.  
Infos are not written to the console, but on the other hand many people think warnings are errors, so a further message type is defined: an important info which is sent to the console.

```

899 \edef\bbl@nulllanguage{\string\language=0}
900 \def\bbl@nocaption{\protect\bbl@nocaption@i}
901 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
902   \global\@namedef{#2}{\textbf{?#1?}}%
903   \@nameuse{#2}%
904   \edef\bbl@tempa{#1}%
905   \bbl@sreplace\bbl@tempa{name}}}%
906   \bbl@warning{%
907     \@backslashchar#1 not set for '\language'. Please,\\%
908     define it after the language has been loaded\\%
909     (typically in the preamble) with:\\%
910     \string\setlocalecaption{\language}{\bbl@tempa}{..}\\%
911     Feel free to contribute on github.com/latex3/babel.\\%
912     Reported}}
913 \def\bbl@tentative{\protect\bbl@tentative@i}
914 \def\bbl@tentative@i#1{%
915   \bbl@warning{%
916     Some functions for '#1' are tentative.\\%
917     They might not work as expected and their behavior\\%
918     could change in the future.\\%
919     Reported}}
920 \def\@nolanerr#1{%
921   \bbl@error
922   {You haven't defined the language '#1' yet.\\%
923     Perhaps you misspelled it or your installation\\%
924     is not complete}%
925   {Your command will be ignored, type <return> to proceed}}
926 \def\@nopatterns#1{%
927   \bbl@warning
928   {No hyphenation patterns were preloaded for\\%
929     the language '#1' into the format.\\%
930     Please, configure your TeX system to add them and\\%
931     rebuild the format. Now I will use the patterns\\%
932     preloaded for \bbl@nulllanguage\space instead}}
933 \let\bbl@usehooks\@gobbletwo
934 \ifx\bbl@onlyswitch\@empty\endinput\fi
935 % Here ended switch.def

```

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

```

936 \ifx\directlua\@undefined\else
937   \ifx\bbl@luapatterns\@undefined
938     \input luababel.def
939   \fi
940 \fi
941 <(Basic macros)>
942 \bbl@trace{Compatibility with language.def}
943 \ifx\bbl@languages\@undefined
944   \ifx\directlua\@undefined
945     \openin1 = language.def % TODO. Remove hardcoded number
946     \ifeof1
947       \closein1

```

```

948     \message{I couldn't find the file language.def}
949   \else
950     \closein1
951     \begingroup
952       \def\addlanguage#1#2#3#4#5{%
953         \expandafter\ifx\csname lang@#1\endcsname\relax\else
954           \global\expandafter\let\csname l@#1\expandafter\endcsname
955             \csname lang@#1\endcsname
956         \fi}%
957       \def\uselanguage#1{%
958         \input language.def
959       \endgroup
960     \fi
961   \fi
962   \chardef\l@english\z@
963 \fi

```

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

```

964 \def\addto#1#2{%
965   \ifx#1\@undefined
966     \def#1{#2}%
967   \else
968     \ifx#1\relax
969       \def#1{#2}%
970     \else
971       {\toks@\expandafter{#1#2}%
972        \xdef#1{\the\toks@}}%
973     \fi
974   \fi}

```

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

```

975 \def\bbl@withactive#1#2{%
976   \begingroup
977   \lccode`~=`#2\relax
978   \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  $\TeX$  macros completely in case their definitions change (they have changed in the past). A macro named `\macro` will be saved new control sequences named `\org@macro`.

```

979 \def\bbl@redefine#1{%
980   \edef\bbl@tempa{\bbl@stripslash#1}%
981   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
982   \expandafter\def\csname\bbl@tempa\endcsname{
983     \@onlypreamble\bbl@redefine

```

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

```

984 \def\bbl@redefine@long#1{%
985   \edef\bbl@tempa{\bbl@stripslash#1}%
986   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
987   \long\expandafter\def\csname\bbl@tempa\endcsname{
988     \@onlypreamble\bbl@redefine@long

```

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

```

989 \def\bbl@redefineroobust#1{%
990   \edef\bbl@tempa{\bbl@stripslash#1}%
991   \bbl@ifunset{\bbl@tempa\space}%
992   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
993    \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
994   {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
995   \@namedef{\bbl@tempa\space}}
996 \@onlypreamble\bbl@redefineroobust

```

### 7.3 Hooks

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

```

997 \bbl@trace{Hooks}
998 \newcommand\AddBabelHook[3][]{%
999   \bbl@ifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}}%
1000 \def\bbl@tempa##1, #3=##2, ##3\@empty{\def\bbl@tempb{##2}}%
1001 \expandafter\bbl@tempa\bbl@evargs, #3=, \@empty
1002 \bbl@ifunset{\bbl@ev@#2@#3@#1}%
1003   {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1004   {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1005 \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1006 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1007 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1008 \def\bbl@usehooks#1#2{%
1009   \ifx\UseHook\undefined\else\UseHook{babel/*/#1}\fi
1010   \def\bbl@elth##1{%
1011     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@#2}}}%
1012     \bbl@cs{ev@#1@}%
1013     \ifx\language\undefined\else % Test required for Plain (?)
1014       \ifx\UseHook\undefined\else\UseHook{babel/\language/#1}\fi
1015       \def\bbl@elth##1{%
1016         \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1@#2}}}%
1017         \bbl@cl{ev@#1@}%
1018       \fi}

```

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

```

1019 \def\bbl@evargs{,% <- don't delete this comma
1020   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1021   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1022   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1023   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1024   beforestart=0,language=2}
1025 \ifx\NewHook\undefined\else
1026   \def\bbl@tempa#1=#2\@{\NewHook{babel/#1}}
1027   \bbl@foreach\bbl@evargs{\bbl@tempa#1\@}
1028 \fi

```

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

```

1029 \bbl@trace{Defining babelensure}
1030 \newcommand\babelensure[2][]{%
1031   \AddBabelHook{babel-ensure}{afterextras}{%

```

```

1032 \ifcase\bb1@select@type
1033 \bb1@cl{e}%
1034 \fi}%
1035 \begingroup
1036 \let\bb1@ens@include\@empty
1037 \let\bb1@ens@exclude\@empty
1038 \def\bb1@ens@fontenc{\relax}%
1039 \def\bb1@tempb##1{%
1040 \ifx\@empty##1\else\noexpand##1\expandafter\bb1@tempb\fi}%
1041 \edef\bb1@tempa{\bb1@tempb#1\@empty}%
1042 \def\bb1@tempb##1=##2\@{\@namedef{bb1@ens@##1}{##2}}%
1043 \bb1@foreach\bb1@tempa{\bb1@tempb##1\@}%
1044 \def\bb1@tempc{\bb1@ensure}%
1045 \expandafter\bb1@add\expandafter\bb1@tempc\expandafter{%
1046 \expandafter{\bb1@ens@include}}%
1047 \expandafter\bb1@add\expandafter\bb1@tempc\expandafter{%
1048 \expandafter{\bb1@ens@exclude}}%
1049 \toks@{\expandafter{\bb1@tempc}}%
1050 \bb1@exp{%
1051 \endgroup
1052 \def\<bb1@e@#2>{\the\toks@{\bb1@ens@fontenc}}}%
1053 \def\bb1@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1054 \def\bb1@tempb##1{% elt for (excluding) \bb1@captionslist list
1055 \ifx##1\undefined % 3.32 - Don't assume the macro exists
1056 \edef##1{\noexpand\bb1@nocaption
1057 {\bb1@stripslash##1}{\language\bb1@stripslash##1}}%
1058 \fi
1059 \ifx##1\@empty\else
1060 \in@{##1}{#2}%
1061 \ifin@ \else
1062 \bb1@ifunset{\bb1@ensure@\language}%
1063 {\bb1@exp{%
1064 \\\DeclareRobustCommand\<bb1@ensure@\language>[1]{%
1065 \\\foreignlanguage{\language}%
1066 {\ifx\relax#3\else
1067 \\\fontencoding{#3}\selectfont
1068 \fi
1069 #####1}}}%
1070 }%
1071 \toks@{\expandafter{##1}}%
1072 \edef##1{%
1073 \bb1@csarg\noexpand{\ensure@\language}%
1074 {\the\toks@}}%
1075 \fi
1076 \expandafter\bb1@tempb
1077 \fi}%
1078 \expandafter\bb1@tempb\bb1@captionslist\today\@empty
1079 \def\bb1@tempa##1{% elt for include list
1080 \ifx##1\@empty\else
1081 \bb1@csarg\in@{\ensure@\language\expandafter}\expandafter{##1}%
1082 \ifin@ \else
1083 \bb1@tempb##1\@empty
1084 \fi
1085 \expandafter\bb1@tempa
1086 \fi}%
1087 \bb1@tempa#1\@empty}
1088 \def\bb1@captionslist{%
1089 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1090 \contentsname\listfigurename\listtablename\indexname\figurename
1091 \tablename\partname\enclname\ccname\headtoname\pagename\seename
1092 \alsoname\proofname\glossaryname}

```

## 7.4 Setting up language files

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

At the start of processing a language definition file we always check the category code of the `@`-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`.

```

1093 \bbl@trace{Macros for setting language files up}
1094 \def\bbl@ldfinit{%
1095   \let\bbl@screset\@empty
1096   \let\BabelStrings\bbl@opt@string
1097   \let\BabelOptions\@empty
1098   \let\BabelLanguages\relax
1099   \ifx\originalTeX\undefined
1100     \let\originalTeX\@empty
1101   \else
1102     \originalTeX
1103   \fi}
1104 \def\LdfInit#1#2{%
1105   \chardef\atcatcode=\catcode`\@
1106   \catcode`\@=11\relax
1107   \chardef\eqcatcode=\catcode`\=
1108   \catcode`\==12\relax
1109   \expandafter\if\expandafter\@backslashchar
1110     \expandafter\@car\string#2\@nil
1111     \ifx#2\undefined\else
1112       \ldf@quit{#1}%
1113     \fi
1114   \else
1115     \expandafter\ifx\csname#2\endcsname\relax\else
1116       \ldf@quit{#1}%
1117     \fi
1118   \fi
1119   \bbl@ldfinit}

```

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

```

1120 \def\ldf@quit#1{%
1121   \expandafter\main@language\expandafter{#1}%
1122   \catcode`\@=\atcatcode \let\atcatcode\relax
1123   \catcode`\==\eqcatcode \let\eqcatcode\relax
1124   \endinput}

```

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

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

```

1125 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1126   \bbl@afterlang
1127   \let\bbl@afterlang\relax

```

```

1128 \let\BabelModifiers\relax
1129 \let\bbl@screset\relax}%
1130 \def\ldf@finish#1{%
1131   \loadlocalcfg{#1}%
1132   \bbl@afterldf{#1}%
1133   \expandafter\main@language\expandafter{#1}%
1134   \catcode`\@=\atcatcode \let\atcatcode\relax
1135   \catcode`\==\eqcatcode \let\eqcatcode\relax}

```

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

```

1136 \@onlypreamble\LdfInit
1137 \@onlypreamble\ldf@quit
1138 \@onlypreamble\ldf@finish

```

`\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.

```

1139 \def\main@language#1{%
1140   \def\bbl@main@language{#1}%
1141   \let\language\bbl@main@language % TODO. Set locale name
1142   \bbl@id@assign
1143   \bbl@patterns{\language}

```

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

```

1144 \def\bbl@beforestart{%
1145   \def\@nolanerr##1{%
1146     \bbl@warning{Undefined language '##1' in aux.\\Reported}}%
1147   \bbl@usehooks{beforestart}{}%
1148   \global\let\bbl@beforestart\relax}
1149 \AtBeginDocument{%
1150   {\@nameuse{bbl@beforestart}}% Group!
1151   \if@filesw
1152     \providecommand\babel@aux[2]{}%
1153     \immediate\write\@mainaux{%
1154       \string\providecommand\string\babel@aux[2]{}%
1155       \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1156     \fi
1157   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1158   \ifbbl@single % must go after the line above.
1159     \renewcommand\selectlanguage[1]{}%
1160     \renewcommand\foreignlanguage[2]{#2}%
1161     \global\let\babel@aux\@gobbletwo % Also as flag
1162   \fi
1163   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1164 \def\select@language@x#1{%
1165   \ifcase\bbl@select@type
1166     \bbl@ifsamestring\language\language{#1}{\select@language{#1}}%
1167   \else
1168     \select@language{#1}%
1169   \fi}

```

## 7.5 Shorthands

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

```

1170 \bbl@trace{Shorhands}
1171 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1172 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1173 \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\@makeother#1}}%
1174 \ifx\nfss@catcodes\undefined\else % TODO - same for above
1175 \begingroup
1176 \catcode`#1\active
1177 \nfss@catcodes
1178 \ifnum\catcode`#1=\active
1179 \endgroup
1180 \bbl@add\nfss@catcodes{\@makeother#1}%
1181 \else
1182 \endgroup
1183 \fi
1184 \fi}

```

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

```

1185 \def\bbl@remove@special#1{%
1186 \begingroup
1187 \def\x##1##2{\ifnum`#1=##2\noexpand\@empty
1188 \else\noexpand##1\noexpand##2\fi}%
1189 \def\do{\x\do}%
1190 \def\@makeother{\x\@makeother}%
1191 \edef\x{\endgroup
1192 \def\noexpand\dospecials{\dospecials}%
1193 \expandafter\ifx\csname @sanitize\endcsname\relax\else
1194 \def\noexpand\@sanitize{\@sanitize}%
1195 \fi}%
1196 \x}

```

`\initiate@active@char` A language definition file can call this macro to make a character active. This macro takes one argument, the character that is to be made active. When the character was already active this macro does nothing. Otherwise, this macro defines the control sequence `\normal@char⟨char⟩` to expand to the character in its ‘normal state’ and it defines the active character to expand to `\normal@char⟨char⟩` by default (`⟨char⟩` being the character to be made active). Later its definition can be changed to expand to `\active@char⟨char⟩` by calling `\bbl@activate{⟨char⟩}`. For example, to make the double quote character active one could have `\initiate@active@char{"}` in a language definition file. This defines " as `\active@prefix "\active@char` (where the first " is the character with its original catcode, when the shorthand is created, and `\active@char` is a single token). In protected contexts, it expands to `\protect "` or `\noexpand "` (ie, with the original "); otherwise `\active@char` is executed. This macro in turn expands to `\normal@char` in “safe” contexts (eg, `\label`), but `\user@active` in normal “unsafe” ones. The latter search a definition in the user, language and system levels, in this order, but if none is found, `\normal@char` is used. However, a deactivated shorthand (with `\bbl@deactivate` is defined as `\active@prefix "\normal@char`. The following macro is used to define shorthands in the three levels. It takes 4 arguments: the (string’ed) character, `\<level>@group`, `<level>@active` and `<next-level>@active` (except in system).

```

1197 \def\bbl@active@def#1#2#3#4{%
1198 \@namedef{#3#1}{%
1199 \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1200 \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1201 \else
1202 \bbl@afterfi\csname#2@sh@#1\endcsname
1203 \fi}%

```

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

```

1204 \long\@namedef{#3@arg#1}##1{%
1205 \expandafter\ifx\csname#2@sh@#1@string##1\endcsname\relax
1206 \bbl@afterelse\csname#4#1\endcsname##1%
1207 \else

```



```

1208      \bbl@afterfi\csname#2@sh@#1@\string##1\endcsname
1209      \fi}}%

```

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

```

1210 \def\initiate@active@char#1{%
1211   \bbl@ifunset{active@char\string#1}%
1212   {\bbl@withactive
1213    {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1214   {}}

```

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

```

1215 \def\@initiate@active@char#1#2#3{%
1216   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1217   \ifx#1\undefined
1218     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\undefined}}%
1219   \else
1220     \bbl@csarg\let{oridef@#2}#1%
1221     \bbl@csarg\edef{oridef@#2}{%
1222       \let\noexpand#1%
1223       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1224   \fi

```

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

```

1225   \ifx#1#3\relax
1226     \expandafter\let\csname normal@char#2\endcsname#3%
1227   \else
1228     \bbl@info{Making #2 an active character}%
1229     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1230     \@namedef{normal@char#2}{%
1231       \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
1232   \else
1233     \@namedef{normal@char#2}{#3}%
1234   \fi

```

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

```

1235   \bbl@restoreactive{#2}%
1236   \AtBeginDocument{%
1237     \catcode`#2\active
1238     \if@files@w
1239       \immediate\write\@mainaux{\catcode`\string#2\active}%
1240     \fi}%
1241   \expandafter\bbl@add@special\csname#2\endcsname
1242   \catcode`#2\active
1243   \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>).

```

1244   \let\bbl@tempa\@firstoftwo

```

```

1245 \if\string^#2%
1246 \def\bbl@tempa{\noexpand\textormath}%
1247 \else
1248 \ifx\bbl@mathnormal\@undefined\else
1249 \let\bbl@tempa\bbl@mathnormal
1250 \fi
1251 \fi
1252 \expandafter\edef\csname active@char#2\endcsname{%
1253 \bbl@tempa
1254 {\noexpand\if@safe@actives
1255 \noexpand\expandafter
1256 \expandafter\noexpand\csname normal@char#2\endcsname
1257 \noexpand\else
1258 \noexpand\expandafter
1259 \expandafter\noexpand\csname bbl@doactive#2\endcsname
1260 \noexpand\fi}%
1261 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1262 \bbl@csarg\edef{doactive#2}{%
1263 \expandafter\noexpand\csname user@active#2\endcsname}%

```

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

`\active@prefix<char>\normal@char<char>`

(where `\active@char<char>` is *one* control sequence!).

```

1264 \bbl@csarg\edef{active@#2}{%
1265 \noexpand\active@prefix\noexpand#1%
1266 \expandafter\noexpand\csname active@char#2\endcsname}%
1267 \bbl@csarg\edef{normal@#2}{%
1268 \noexpand\active@prefix\noexpand#1%
1269 \expandafter\noexpand\csname normal@char#2\endcsname}%
1270 \bbl@ncarg\let#1\bbl@normal@#2}%

```

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

```

1271 \bbl@active@def#2\user@group{user@active}{language@active}%
1272 \bbl@active@def#2\language@group{language@active}{system@active}%
1273 \bbl@active@def#2\system@group{system@active}{normal@char}%

```

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

```

1274 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1275 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1276 \expandafter\edef\csname\user@group @sh#2@\string\protect@\endcsname
1277 {\expandafter\noexpand\csname user@active#2\endcsname}%

```

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

```

1278 \if\string'#2%
1279 \let\prim@s\bbl@prim@s
1280 \let\active@math@prime#1%
1281 \fi
1282 \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}

```

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

```

1283 <{*More package options}> ≡
1284 \DeclareOption{math=active}{ }

```

```

1285 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1286 <\/More package options>

```

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

```

1287 \@ifpackagewith{babel}{KeepShorthandsActive}%
1288 {\let\bbl@restoreactive\@gobble}%
1289 {\def\bbl@restoreactive#1{%
1290   \bbl@exp{%
1291     \\\AfterBabelLanguage\\CurrentOption
1292     {\catcode`#1=\the\catcode`#1\relax}%
1293     \\\AtEndOfPackage
1294     {\catcode`#1=\the\catcode`#1\relax}}}%
1295   \AtEndOfPackage{\let\bbl@restoreactive\@gobble}}

```

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

```

1296 \def\bbl@sh@select#1#2{%
1297   \expandafter\ifx\csname#1@sh#2@sel\endcsname\relax
1298     \bbl@afterelse\bbl@scndcs
1299   \else
1300     \bbl@afterfi\csname#1@sh#2@sel\endcsname
1301   \fi}

```

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

```

1302 \begingroup
1303 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct? Only Plain?
1304 {\gdef\active@prefix#1{%
1305   \ifx\protect\@typeset@protect
1306     \else
1307       \ifx\protect\@unexpandable@protect
1308         \noexpand#1%
1309       \else
1310         \protect#1%
1311       \fi
1312       \expandafter\@gobble
1313     \fi}}
1314 {\gdef\active@prefix#1{%
1315   \ifincsname
1316     \string#1%
1317     \expandafter\@gobble
1318   \else
1319     \ifx\protect\@typeset@protect
1320     \else
1321       \ifx\protect\@unexpandable@protect
1322         \noexpand#1%
1323       \else
1324         \protect#1%
1325       \fi
1326       \expandafter\expandafter\expandafter\@gobble
1327     \fi
1328   \fi}}
1329 \endgroup

```

`\if@safe@actives` 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<char>`.

```
1330 \newif\if@safe@actives
1331 \@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.

```
1332 \def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}
```

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

```
1333 \chardef\bbl@activated\z@
1334 \def\bbl@activate#1{%
1335   \chardef\bbl@activated\@ne
1336   \bbl@withactive{\expandafter\let\expandafter}#1%
1337   \csname bbl@active@\string#1\endcsname}
1338 \def\bbl@deactivate#1{%
1339   \chardef\bbl@activated\tw@
1340   \bbl@withactive{\expandafter\let\expandafter}#1%
1341   \csname bbl@normal@\string#1\endcsname}
```

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

```
\bbl@scndcs
1342 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1343 \def\bbl@scndcs#1#2{\csname#2\endcsname}
```

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

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

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

```
1344 \def\babel@texpdf#1#2#3#4{%
1345   \ifx\texorpdfstring\undefined
1346     \textormath{#1}{#3}%
1347   \else
1348     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1349     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1350   \fi}
1351 %
1352 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1353 \def\@decl@short#1#2#3\@nil#4{%
1354   \def\bbl@tempa{#3}%
1355   \ifx\bbl@tempa\@empty
1356     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1357     \bbl@ifunset{#1@sh@\string#2@}{}%
1358     {\def\bbl@tempa{#4}%
1359      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1360      \else
1361        \bbl@info
1362        {Redefining #1 shorthand \string#2\%
1363         in language \CurrentOption}%
1364      \fi}%
1365   \@namedef{#1@sh@\string#2@}{#4}%
```

```

1366 \else
1367 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bb1@firstcs
1368 \bb1@ifunset{#1@sh@\string#2@\string#3@}{}%
1369 {\def\bb1@tempa{#4}%
1370 \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bb1@tempa
1371 \else
1372 \bb1@info
1373 {Redefining #1 shorthand \string#2\string#3\\%
1374 in language \CurrentOption}%
1375 \fi}%
1376 \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1377 \fi}

```

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

```

1378 \def\textormath{%
1379 \ifmmode
1380 \expandafter\@secondoftwo
1381 \else
1382 \expandafter\@firstoftwo
1383 \fi}

```

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

```

1384 \def\user@group{user}
1385 \def\language@group{english} % TODO. I don't like defaults
1386 \def\system@group{system}

```

`\usesshorthands` 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.

```

1387 \def\usesshorthands{%
1388 \@ifstar\bb1@usesesh{s}\bb1@usesesh{x}}%
1389 \def\bb1@usesesh{s#1}{%
1390 \bb1@useseshx
1391 {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb1@activate{#1}}}%
1392 {#1}}
1393 \def\bb1@usesesh{x#1#2}{%
1394 \bb1@ifshorthand{#2}%
1395 {\def\user@group{user}%
1396 \initiate@active@char{#2}%
1397 #1%
1398 \bb1@activate{#2}}%
1399 {\bb1@error
1400 {I can't declare a shorthand turned off (\string#2)}
1401 {Sorry, but you can't use shorthands which have been\\%
1402 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 `\bb1@set@user@generic`); we make also sure `{}` and `\protect` are taken into account in this new top level.

```

1403 \def\user@language@group{user@\language@group}
1404 \def\bb1@set@user@generic#1#2{%
1405 \bb1@ifunset{user@generic@active#1}%
1406 {\bb1@active@def#1\user@language@group{user@active}{user@generic@active}%
1407 \bb1@active@def#1\user@group{user@generic@active}{language@active}%
1408 \expandafter\edef\csname#2@sh@#1@\endcsname{%
1409 \expandafter\noexpand\csname normal@char#1\endcsname}%
1410 \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%

```

```

1411 \expandafter\noexpand\csname user@active#1\endcsname}}%
1412 \@empty}
1413 \newcommand\defineshorthand[3][user]{%
1414 \edef\bbl@tempa{\zap@space#1 \@empty}%
1415 \bbl@for\bbl@tempb\bbl@tempa{%
1416 \if*\expandafter\@car\bbl@tempb\@nil
1417 \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1418 \@expandtwoargs
1419 \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1420 \fi
1421 \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].

```

1422 \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`".

```

1423 \def\aliasshorthand#1#2{%
1424 \bbl@ifshorthand{#2}%
1425 {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1426 \ifx\document\@notprerr
1427 \@notshorthand{#2}%
1428 \else
1429 \initiate@active@char{#2}%
1430 \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1431 \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1432 \bbl@activate{#2}%
1433 \fi
1434 \fi}%
1435 {\bbl@error
1436 {Cannot declare a shorthand turned off (\string#2)}
1437 {Sorry, but you cannot use shorthands which have been\\%
1438 turned off in the package options}}}

```

`\@notshorthand`

```

1439 \def\@notshorthand#1{%
1440 \bbl@error{%
1441 The character '\string #1' should be made a shorthand character;\\%
1442 add the command \string\usesshorthands\string{#1\string} to
1443 the preamble.\\%
1444 I will ignore your instruction}%
1445 {You may proceed, but expect unexpected results}}

```

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

```

1446 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1447 \DeclareRobustCommand*\shorthandoff{%
1448 \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1449 \def\bbl@shorthandoff#1#2{\bbl@switch@sh#1#2\@nnil}

```

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

```

1450 \def\bbl@switch@sh#1#2{%
1451 \ifx#2\@nnil\else

```

```

1452 \bbl@ifunset{bbl@active@\string#2}%
1453   {\bbl@error
1454     {I can't switch '\string#2' on or off--not a shorthand}%
1455     {This character is not a shorthand. Maybe you made\%
1456       a typing mistake? I will ignore your instruction.}}%
1457   {\ifcase#1%   off, on, off*
1458     \catcode`#2\relax
1459   \or
1460     \catcode`#2\active
1461     \bbl@ifunset{bbl@shdef@\string#2}%
1462     {}%
1463     {\bbl@withactive{\expandafter\let\expandafter}%#2%
1464       \csname bbl@shdef@\string#2\endcsname
1465       \bbl@csarg\let{shdef@\string#2}\relax}%
1466     \ifcase\bbl@activated\or
1467       \bbl@activate{#2}%
1468     \else
1469       \bbl@deactivate{#2}%
1470     \fi
1471   \or
1472     \bbl@ifunset{bbl@shdef@\string#2}%
1473     {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}%#2%
1474     {}%
1475     \csname bbl@oricat@\string#2\endcsname
1476     \csname bbl@oridef@\string#2\endcsname
1477     \fi}%
1478   \bbl@afterfi\bbl@switch@sh#1%
1479 \fi}

```

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

```

1480 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1481 \def\bbl@putsh#1{%
1482   \bbl@ifunset{bbl@active@\string#1}%
1483   {\bbl@putsh@i#1\@empty\@nnil}%
1484   {\csname bbl@active@\string#1\endcsname}}
1485 \def\bbl@putsh@i#1#2\@nnil{%
1486   \csname\language@group @sh@\string#1@%
1487     \ifx\@empty#2\else\string#2@\fi\endcsname}
1488 \ifx\bbl@opt@shorthands\@nnil\else
1489   \let\bbl@s@initiate@active@char\initiate@active@char
1490   \def\initiate@active@char#1{%
1491     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1492   \let\bbl@s@switch@sh\bbl@switch@sh
1493   \def\bbl@switch@sh#1#2{%
1494     \ifx#2\@nnil\else
1495       \bbl@afterfi
1496       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1497     \fi}
1498   \let\bbl@s@activate\bbl@activate
1499   \def\bbl@activate#1{%
1500     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1501   \let\bbl@s@deactivate\bbl@deactivate
1502   \def\bbl@deactivate#1{%
1503     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1504 \fi

```

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

```

1505 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{bbl@active@\string#1}{#3}{#2}}

```

`\bbl@prim@s` One of the internal macros that are involved in substituting `\prime` for each right quote in mathmode is `\prim@s`. This checks if the next character is a right quote. When the right quote is active, the definition of this macro needs to be adapted to look also for an active right quote; the hat could be active, too.

```

1506 \def\bbl@prim@s{%
1507   \prime\futurelet\@let@token\bbl@pr@m@s}
1508 \def\bbl@if@primes#1#2{%
1509   \ifx#1\@let@token
1510     \expandafter\@firstoftwo
1511   \else\ifx#2\@let@token
1512     \bbl@afterelse\expandafter\@firstoftwo
1513   \else
1514     \bbl@afterfi\expandafter\@secondoftwo
1515   \fi\fi}
1516 \begingroup
1517   \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
1518   \catcode`\'=12 \catcode`\"=\active \lccode`\"=\'
1519   \lowercase{%
1520     \gdef\bbl@pr@m@s{%
1521       \bbl@if@primes" '%
1522       \pr@@@s
1523       {\bbl@if@primes*\^pr@@@t\egroup}}
1524 \endgroup

```

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

```

1525 \initiate@active@char{~}
1526 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1527 \bbl@activate{~}

```

\OT1dqpos 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.

```

1528 \expandafter\def\csname OT1dqpos\endcsname{127}
1529 \expandafter\def\csname T1dqpos\endcsname{4}

```

When the macro \f@encoding is undefined (as it is in plain T<sub>E</sub>X) we define it here to expand to OT1

```

1530 \ifx\f@encoding\undefined
1531   \def\f@encoding{OT1}
1532 \fi

```

## 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.

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

```

1533 \bbl@trace{Language attributes}
1534 \newcommand\languageattribute[2]{%
1535   \def\bbl@tempc{#1}%
1536   \bbl@fixname\bbl@tempc
1537   \bbl@iflanguage\bbl@tempc{%
1538     \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.

```

1539     \ifx\bbl@known@attribs\undefined
1540       \in@false
1541     \else
1542       \bbl@xin@{\bbl@tempc-##1,}{\bbl@known@attribs,}%

```



```

1543 \fi
1544 \ifin@
1545 \bbl@warning{%
1546     You have more than once selected the attribute '##1'\%
1547     for language #1. Reported}%
1548 \else

```

When we end up here the attribute is not selected before. So, we add it to the list of selected attributes and execute the associated T<sub>E</sub>X-code.

```

1549 \bbl@exp{%
1550     \\\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
1551 \edef\bbl@tempa{\bbl@tempc-##1}%
1552 \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1553 {\csname\bbl@tempc @attr##1\endcsname}%
1554 {\@attrerr{\bbl@tempc}{##1}}%
1555 \fi}}
1556 \@onlypreamble\languageattribute

```

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

```

1557 \newcommand*{\@attrerr}[2]{%
1558 \bbl@error
1559 {The attribute #2 is unknown for language #1.}%
1560 {Your command will be ignored, type <return> to proceed}}

```

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

```

1561 \def\bbl@declare@ttribute#1#2#3{%
1562 \bbl@xin@{, #2, }{\, \BabelModifiers,}%
1563 \ifin@
1564 \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1565 \fi
1566 \bbl@add@list\bbl@attributes{#1-#2}%
1567 \expandafter\def\csname#1@attr@#2\endcsname{#3}}

```

**\bbl@ifattributeset** This internal macro has 4 arguments. It can be used to interpret T<sub>E</sub>X 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.

```

1568 \def\bbl@ifattributeset#1#2#3#4{%
1569 \ifx\bbl@known@attribs\undefined
1570 \in@false
1571 \else
1572 \bbl@xin@{, #1-#2, }{\, \bbl@known@attribs,}%
1573 \fi
1574 \ifin@
1575 \bbl@afterelse#3%
1576 \else
1577 \bbl@afterfi#4%
1578 \fi}

```

**\bbl@ifknown@ttrib** An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the T<sub>E</sub>X-code to be executed when the attribute is known and the T<sub>E</sub>X-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.

```

1579 \def\bbl@ifknown@ttrib#1#2{%
1580 \let\bbl@tempa\@secondoftwo
1581 \bbl@loopx\bbl@tempb{#2}{%
1582 \expandafter\in@\expandafter{\expandafter, \bbl@tempb, }{, #1,}%
1583 \ifin@

```

```

1584     \let\bbl@tempa\@firstoftwo
1585     \else
1586     \fi}%
1587     \bbl@tempa}

```

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

```

1588 \def\bbl@clear@ttribs{%
1589     \ifx\bbl@attributes\@undefined\else
1590     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1591         \expandafter\bbl@clear@ttrib\bbl@tempa.
1592     }%
1593     \let\bbl@attributes\@undefined
1594     \fi}
1595 \def\bbl@clear@ttrib#1-#2.{%
1596     \expandafter\let\csname#1@attr#2\endcsname\@undefined}
1597 \AtBeginDocument{\bbl@clear@ttribs}

```

## 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 `\originalTeX`). Note undefined macros are not undefined any more when saved – they are `\relax`'ed.

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

```

1598 \bbl@trace{Macros for saving definitions}
1599 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

1600 \newcount\babel@savecnt
1601 \babel@beginsave

```

`\babel@save` The macro `\babel@save<csname>` saves the current meaning of the control sequence `<csname>` to `\originalTeX`<sup>32</sup>. To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to `\originalTeX` and the counter is incremented. The macro `\babel@savevariable<variable>` saves the value of the variable. `<variable>` can be anything allowed after the `\the` primitive.

```

1602 \def\babel@save#1{%
1603     \expandafter\let\csname babel@\number\babel@savecnt\endcsname#1\relax
1604     \toks@\expandafter{\originalTeX\let#1=}%
1605     \bbl@exp{%
1606         \def\\originalTeX{\the\toks@\<babel@\number\babel@savecnt>\relax}}%
1607     \advance\babel@savecnt\@ne}
1608 \def\babel@savevariable#1{%
1609     \toks@\expandafter{\originalTeX #1=}%
1610     \bbl@exp{\def\\originalTeX{\the\toks@\the#1\relax}}}

```

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

```

1611 \def\bbl@frenchspacing{%
1612     \ifnum\the\sfcodes\`.\=@m
1613     \let\bbl@nonfrenchspacing\relax
1614     \else
1615     \frenchspacing

```

<sup>32</sup>`\originalTeX` has to be expandable, i.e. you shouldn't let it to `\relax`.

```

1616 \let\bbl@nonfrenchspacing\nonfrenchspacing
1617 \fi}
1618 \let\bbl@nonfrenchspacing\nonfrenchspacing
1619 \let\bbl@elt\relax
1620 \edef\bbl@fs@chars{%
1621 \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1622 \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1623 \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1624 \def\bbl@pre@fs{%
1625 \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1626 \edef\bbl@save@sfcodes{\bbl@fs@chars}%
1627 \def\bbl@post@fs{%
1628 \bbl@save@sfcodes
1629 \edef\bbl@tempa{\bbl@cl{frspc}}}%
1630 \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1631 \if u\bbl@tempa % do nothing
1632 \else\if n\bbl@tempa % non french
1633 \def\bbl@elt##1##2##3{%
1634 \ifnum\sfcode`##1=##2\relax
1635 \babel@savevariable{\sfcode`##1}%
1636 \sfcode`##1=##3\relax
1637 \fi}%
1638 \bbl@fs@chars
1639 \else\if y\bbl@tempa % french
1640 \def\bbl@elt##1##2##3{%
1641 \ifnum\sfcode`##1=##3\relax
1642 \babel@savevariable{\sfcode`##1}%
1643 \sfcode`##1=##2\relax
1644 \fi}%
1645 \bbl@fs@chars
1646 \fi\fi\fi}

```

## 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.

```

1647 \bbl@trace{Short tags}
1648 \def\babeltags#1{%
1649 \edef\bbl@tempa{\zap@space#1 \@empty}%
1650 \def\bbl@tempb##1=##2\@{%
1651 \edef\bbl@tempc{%
1652 \noexpand\newcommand
1653 \expandafter\noexpand\csname ##1\endcsname{%
1654 \noexpand\protect
1655 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1656 \noexpand\newcommand
1657 \expandafter\noexpand\csname text##1\endcsname{%
1658 \noexpand\foreignlanguage{##2}}}%
1659 \bbl@tempc}%
1660 \bbl@for\bbl@tempa\bbl@tempa{%
1661 \expandafter\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.

```

1662 \bbl@trace{Hyphens}
1663 \@onlypreamble\babelhyphenation
1664 \AtEndOfPackage{%
1665 \newcommand\babelhyphenation[2][\@empty]{%

```

```

1666 \ifx\bbl@hyphenation@ \relax
1667 \let\bbl@hyphenation@ \@empty
1668 \fi
1669 \ifx\bbl@hyphlist@ \@empty \else
1670 \bbl@warning{%
1671 You must not intermingle \string\selectlanguage\space and\%
1672 \string\babelhyphenation\space or some exceptions will not\%
1673 be taken into account. Reported}%
1674 \fi
1675 \ifx\@empty#1%
1676 \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1677 \else
1678 \bbl@vforeach{#1}{%
1679 \def\bbl@tempa{##1}%
1680 \bbl@fixname\bbl@tempa
1681 \bbl@iflanguage\bbl@tempa{%
1682 \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1683 \bbl@ifunset{\bbl@hyphenation@\bbl@tempa}%
1684 }{%
1685 {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1686 #2}}}%
1687 \fi}}

```

`\bbl@allowhyphens` This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak \hskip Opt plus Opt`<sup>33</sup>.

```

1688 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1689 \def\bbl@t@one{T1}
1690 \def\allowhyphens{\ifx\cf@encoding\bbl@t@one\else\bbl@allowhyphens\fi}

```

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

```

1691 \newcommand\babellnullhyphen{\char\hyphenchar\font}
1692 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1693 \def\bbl@hyphen{%
1694 \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
1695 \def\bbl@hyphen@i#1#2{%
1696 \bbl@ifunset{\bbl@hy#1#2\@empty}%
1697 {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{\#2}}}%
1698 {\csname bbl@hy#1#2\@empty\endcsname}}

```

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

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

```

1699 \def\bbl@usehyphen#1{%
1700 \leavevmode
1701 \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1702 \nobreak\hskip\z@skip}
1703 \def\bbl@@usehyphen#1{%
1704 \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1705 \def\bbl@hyphenchar{%
1706 \ifnum\hyphenchar\font=\m@ne
1707 \babellnullhyphen
1708 \else
1709 \char\hyphenchar\font
1710 \fi}

```

<sup>33</sup>TeX begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

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

```

1711 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1712 \def\bbl@hy@@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1713 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1714 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1715 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1716 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1717 \def\bbl@hy@repeat{%
1718   \bbl@usehyphen{%
1719     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1720 \def\bbl@hy@@repeat{%
1721   \bbl@usehyphen{%
1722     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1723 \def\bbl@hy@empty{\hskip\z@skip}
1724 \def\bbl@hy@@empty{\discretionary{}{}{}}

```

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

```

1725 \def\bbl@disc#1#2{\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 tool. It makes global a local variable. This is not the best solution, but it works.

```

1726 \bbl@trace{Multiencoding strings}
1727 \def\bbl@toggle#1{\global\let#1#1}

```

The second one. We need to patch `\@uclclist`, but it is done once and only if `\SetCase` is used or if strings are encoded. The code is far from satisfactory for several reasons, including the fact `\@uclclist` is not a list any more. Therefore a package option is added to ignore it. Instead of gobbling the macro getting the next two elements (usually `\reserved@a`), we pass it as argument to `\bbl@uclc`. The parser is restarted inside `\langle lang\rangle\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).

```

1728 \@ifpackagewith{babel}{nocase}%
1729   {\let\bbl@patchuclc\relax}%
1730   {\def\bbl@patchuclc{%
1731     \global\let\bbl@patchuclc\relax
1732     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1733     \gdef\bbl@uclc##1{%
1734       \let\bbl@encoded\bbl@encoded@uclc
1735       \bbl@ifunset{\language @bbl@uclc}% and resumes it
1736       {##1}%
1737       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1738         \csname\language @bbl@uclc\endcsname}%
1739       {\bbl@tolower\empty}{\bbl@toupper\empty}}}%
1740   \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
1741   \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}
1742 % A temporary hack:
1743 \ifx\BabelCaseHack\undefined
1744 \AtBeginDocument{%
1745   \bbl@xin@{\string\@uclclist}%
1746   {\bbl@carg\meaning\MakeUppercase }}%

```

```

1747 \ifin@else
1748 \chardef\bbl@ulflag\z@
1749 \bbl@ncarg\let\bbl@newuc{MakeUppercase}%
1750 \protected\@namedef{MakeUppercase}#1{%
1751 \chardef\bbl@ulflag@ne
1752 \ifx\bbl@ucllc\@undefined
1753 \bbl@newuc{#1}%
1754 \else
1755 \bbl@ifunset{\language @bbl@ucllc}%
1756 {\bbl@newuc{#1}}%
1757 {\def\reserved@a##1##2{\let##1##2\reserved@a}%
1758 \bbl@ucllc\reserved@a\reserved@b{\reserved@b\@gobble}%
1759 \protected@edef\reserved@a{\bbl@newuc{#1}}% Pre-expand
1760 \reserved@a}%
1761 \fi}}%
1762 \bbl@ncarg\let\bbl@newlc{MakeLowercase}%
1763 \protected\@namedef{MakeLowercase}#1{%
1764 \chardef\bbl@ulflag\tw@
1765 \ifx\bbl@ucllc\@undefined
1766 \bbl@newlc{#1}%
1767 \else
1768 \bbl@ifunset{\language @bbl@ucllc}%
1769 {\bbl@newlc{#1}}%
1770 {\def\reserved@a##1##2{\let##2##1\reserved@a}%
1771 \bbl@ucllc\reserved@a\reserved@b{\reserved@b\@gobble}%
1772 \protected@edef\reserved@a{\bbl@newlc{#1}}% Pre-expand
1773 \reserved@a}%
1774 \fi}}%
1775 \def\bbl@cased{%
1776 \ifcase\bbl@ulflag
1777 \expandafter\@firstofone
1778 \or
1779 \expandafter\MakeUppercase
1780 \or
1781 \expandafter\MakeLowercase
1782 \fi}%
1783 \fi}
1784 \fi

1785 <<{*More package options}>> ≡
1786 \DeclareOption{nocase}{}
1787 <</More package options>>

```

The following package options control the behavior of \SetString.

```

1788 <<{*More package options}>> ≡
1789 \let\bbl@opt@strings\@nnil % accept strings=value
1790 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1791 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1792 \def\BabelStringsDefault{generic}
1793 <</More package options>>

```

**Main command** This is the main command. With the first use it is redefined to omit the basic setup in subsequent blocks. We make sure strings contain actual letters in the range 128-255, not active characters.

```

1794 \@onlypreamble\StartBabelCommands
1795 \def\StartBabelCommands{%
1796 \begingroup
1797 \@tempcnta="7F
1798 \def\bbl@tempa{%
1799 \ifnum\@tempcnta>"FF\else
1800 \catcode\@tempcnta=11
1801 \advance\@tempcnta@ne
1802 \expandafter\bbl@tempa

```

```

1803 \fi}%
1804 \bbl@tempa
1805 <<Macros local to BabelCommands>>
1806 \def\bbl@provstring##1##2{%
1807 \providecommand##1{##2}%
1808 \bbl@tglobal##1}%
1809 \global\let\bbl@scafter\@empty
1810 \let\StartBabelCommands\bbl@startcmds
1811 \ifx\BabelLanguages\relax
1812 \let\BabelLanguages\CurrentOption
1813 \fi
1814 \begingroup
1815 \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1816 \StartBabelCommands}
1817 \def\bbl@startcmds{%
1818 \ifx\bbl@screset\@nnil\else
1819 \bbl@usehooks{stopcommands}{}%
1820 \fi
1821 \endgroup
1822 \begingroup
1823 \@ifstar
1824 {\ifx\bbl@opt@strings\@nnil
1825 \let\bbl@opt@strings\BabelStringsDefault
1826 \fi
1827 \bbl@startcmds@i}%
1828 \bbl@startcmds@i}
1829 \def\bbl@startcmds@i#1#2{%
1830 \edef\bbl@L{\zap@space#1 \@empty}%
1831 \edef\bbl@G{\zap@space#2 \@empty}%
1832 \bbl@startcmds@ii}
1833 \let\bbl@startcommands\StartBabelCommands

```

Parse the encoding info to get the label, input, and font parts.

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

```

1834 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1835 \let\SetString\gobbletwo
1836 \let\bbl@stringdef\gobbletwo
1837 \let\AfterBabelCommands\gobble
1838 \ifx\@empty#1%
1839 \def\bbl@sc@label{generic}%
1840 \def\bbl@encstring##1##2{%
1841 \ProvideTextCommandDefault##1{##2}%
1842 \bbl@tglobal##1%
1843 \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%
1844 \let\bbl@sctest\in@true
1845 \else
1846 \let\bbl@sc@charset\space % <- zapped below
1847 \let\bbl@sc@fontenc\space % <- " "
1848 \def\bbl@tempa##1=##2\@nil{%
1849 \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1850 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1851 \def\bbl@tempa##1 ##2{% space -> comma
1852 ##1%
1853 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1854 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1855 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%

```

```

1856 \edef\bb1@sc@charset{\expandafter\zap@space\bb1@sc@charset\@empty}%
1857 \def\bb1@encstring##1##2{%
1858   \bb1@foreach\bb1@sc@fontenc{%
1859     \bb1@ifunset{T@###1}%
1860     {}%
1861     {\ProvideTextCommand##1{###1}{##2}%
1862       \bb1@toglobal##1%
1863       \expandafter
1864       \bb1@toglobal\csname###1\string##1\endcsname}}}%
1865 \def\bb1@sctest{%
1866   \bb1@xin@{\bb1@opt@strings,}{\bb1@sc@label,\bb1@sc@fontenc,}%
1867 \fi
1868 \ifx\bb1@opt@strings\@nnil      % ie, no strings key -> defaults
1869 \else\ifx\bb1@opt@strings\relax % ie, strings=encoded
1870   \let\AfterBabelCommands\bb1@aftercmds
1871   \let\SetString\bb1@setstring
1872   \let\bb1@stringdef\bb1@encstring
1873 \else      % ie, strings=value
1874 \bb1@sctest
1875 \ifin@
1876   \let\AfterBabelCommands\bb1@aftercmds
1877   \let\SetString\bb1@setstring
1878   \let\bb1@stringdef\bb1@provstring
1879 \fi\fi\fi
1880 \bb1@scswitch
1881 \ifx\bb1@G\@empty
1882   \def\SetString##1##2{%
1883     \bb1@error{Missing group for string \string##1}%
1884     {You must assign strings to some category, typically\\
1885      captions or extras, but you set none}}%
1886 \fi
1887 \ifx\@empty#1%
1888   \bb1@usehooks{defaultcommands}{}%
1889 \else
1890   \@expandtwoargs
1891   \bb1@usehooks{encodedcommands}{\bb1@sc@charset}{\bb1@sc@fontenc}}%
1892 \fi}

```

There are two versions of `\bb1@scswitch`. The first version is used when `ldfs` are read, and it makes sure `\group` *(language)* is reset, but only once (`\bb1@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 `\bb1@forlang` loops `\bb1@L` but its body is executed only if the value is in `\BabelLanguages` (inside `babel`) or `\date` *(language)* is defined (after `babel` has been loaded). There are also two version of `\bb1@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).

```

1893 \def\bb1@forlang#1#2{%
1894   \bb1@for#1\bb1@L{%
1895     \bb1@xin@{, #1, }{\BabelLanguages,}%
1896     \ifin#2\relax\fi}}
1897 \def\bb1@scswitch{%
1898   \bb1@forlang\bb1@tempa{%
1899     \ifx\bb1@G\@empty\else
1900       \ifx\SetString@gobbletwo\else
1901         \edef\bb1@GL{\bb1@G\bb1@tempa}%
1902         \bb1@xin@{\bb1@GL,}{\bb1@screset,}%
1903       \ifin\@else
1904         \global\expandafter\let\csname\bb1@GL\endcsname\@undefined
1905       \xdef\bb1@screset{\bb1@screset,\bb1@GL}%
1906     \fi
1907   \fi
1908 \fi}}
1909 \AtEndOfPackage{%

```



```

1910 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\#2}}}%
1911 \let\bbl@scswitch\relax}
1912 \@onlypreamble\EndBabelCommands
1913 \def\EndBabelCommands{%
1914   \bbl@usehooks{stopcommands}{}%
1915   \endgroup
1916   \endgroup
1917   \bbl@scafter}
1918 \let\bbl@endcommands\EndBabelCommands

```

Now we define commands to be used inside \StartBabelCommands.

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

```

1919 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1920   \bbl@forlang\bbl@tempa{%
1921     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1922     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1923       {\bbl@exp{%
1924         \global\bbbl@add<\bbl@G\bbl@tempa>{\bbbl@scset\#1<\bbl@LC>}}}%
1925       }%
1926   \def\BabelString{#2}%
1927   \bbl@usehooks{stringprocess}{}%
1928   \expandafter\bbl@stringdef
1929   \csname\bbl@LC\expandafter\endcsname\expandafter{\BabelString}}

```

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

```

1930 \ifx\bbl@opt@strings\relax
1931   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1932   \bbl@patchuclc
1933   \let\bbl@encoded\relax
1934   \def\bbl@encoded@uclc#1{%
1935     \@inmathwarn#1%
1936     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1937       \expandafter\ifx\csname ?\string#1\endcsname\relax
1938         \TextSymbolUnavailable#1%
1939       \else
1940         \csname ?\string#1\endcsname
1941       \fi
1942     \else
1943       \csname\cf@encoding\string#1\endcsname
1944     \fi}
1945 \else
1946   \def\bbl@scset#1#2{\def#1{\#2}}
1947 \fi

```

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

```

1948 <(*Macros local to BabelCommands)> ≡
1949 \def\SetStringLoop##1##2{%
1950   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1951   \count@z@
1952   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1953     \advance\count@\@ne
1954     \toks@\expandafter{\bbl@tempa}%
1955     \bbl@exp{%
1956       \SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%

```

```

1957 \count@=\the\count@\relax}}}%
1958 <\/Macros local to BabelCommands>

```

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

```

1959 \def\bbl@aftercmds#1{%
1960 \toks@\expandafter{\bbl@scafter#1}%
1961 \xdef\bbl@scafter{\the\toks@}}

```

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

```

1962 <(*Macros local to BabelCommands)> ≡
1963 \newcommand\SetCase[3][{}]{%
1964 \bbl@patchuclc
1965 \bbl@forlang\bbl@tempa{%
1966 \bbl@carg\bbl@encstring{\bbl@tempa @bbl@uclc}{\bbl@tempa##1}%
1967 \bbl@carg\bbl@encstring{\bbl@tempa @bbl@uc}{##2}%
1968 \bbl@carg\bbl@encstring{\bbl@tempa @bbl@lc}{##3}}}%
1969 <\/Macros local to BabelCommands>

```

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

```

1970 <(*Macros local to BabelCommands)> ≡
1971 \newcommand\SetHyphenMap[1]{%
1972 \bbl@forlang\bbl@tempa{%
1973 \expandafter\bbl@stringdef
1974 \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1975 <\/Macros local to BabelCommands>

```

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

```

1976 \newcommand\BabelLower[2]{% one to one.
1977 \ifnum\lccode#1=#2\else
1978 \babel@savevariable{\lccode#1}%
1979 \lccode#1=#2\relax
1980 \fi}
1981 \newcommand\BabelLowerMM[4]{% many-to-many
1982 \@tempcnta=#1\relax
1983 \@tempcntb=#4\relax
1984 \def\bbl@tempa{%
1985 \ifnum\@tempcnta>#2\else
1986 \expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1987 \advance\@tempcnta#3\relax
1988 \advance\@tempcntb#3\relax
1989 \expandafter\bbl@tempa
1990 \fi}%
1991 \bbl@tempa}
1992 \newcommand\BabelLowerMO[4]{% many-to-one
1993 \@tempcnta=#1\relax
1994 \def\bbl@tempa{%
1995 \ifnum\@tempcnta>#2\else
1996 \expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1997 \advance\@tempcnta#3
1998 \expandafter\bbl@tempa
1999 \fi}%
2000 \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

2001 <(*More package options)> ≡
2002 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
2003 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
2004 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}

```

```

2005 \DeclareOption{hyphenmap=other}{\chardef\bb1@opt@hyphenmap\thr@@}
2006 \DeclareOption{hyphenmap=other*}{\chardef\bb1@opt@hyphenmap4\relax}
2007 <\/More package options>

```

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

```

2008 \AtEndOfPackage{%
2009   \ifx\bb1@opt@hyphenmap\undefined
2010     \bb1@xin@{,}{\bb1@language@opts}%
2011     \chardef\bb1@opt@hyphenmap\ifin@4\else\@ne\fi
2012   \fi}

```

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

```

2013 \newcommand\setlocalecaption{% TODO. Catch typos.
2014   \@ifstar\bb1@setcaption@s\bb1@setcaption@x}
2015 \def\bb1@setcaption@x#1#2#3{% language caption-name string
2016   \bb1@trim@def\bb1@tempa{#2}%
2017   \bb1@xin@{.template}{\bb1@tempa}%
2018   \ifin@
2019     \bb1@ini@captions@template{#3}{#1}%
2020   \else
2021     \edef\bb1@tempd{%
2022       \expandafter\expandafter\expandafter
2023       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2024     \bb1@xin@
2025       {\expandafter\string\csname #2name\endcsname}%
2026       {\bb1@tempd}%
2027     \ifin@ % Renew caption
2028       \bb1@xin@{\string\bb1@scset}{\bb1@tempd}%
2029     \ifin@
2030       \bb1@exp{%
2031         \\bb1@ifsamestring{\bb1@tempa}{\language}%
2032         {\bb1@scset\<#2name>\<#1#2name>}%
2033         {}}%
2034       \else % Old way converts to new way
2035         \bb1@ifunset{#1#2name}%
2036         {\bb1@exp{%
2037           \\bb1@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2038           \\bb1@ifsamestring{\bb1@tempa}{\language}%
2039           {\def\<#2name>{\<#1#2name>}}%
2040           {}}}%
2041         {}%
2042       \fi
2043     \else
2044       \bb1@xin@{\string\bb1@scset}{\bb1@tempd}% New
2045       \ifin@ % New way
2046         \bb1@exp{%
2047           \\bb1@add\<captions#1>{\bb1@scset\<#2name>\<#1#2name>}%
2048           \\bb1@ifsamestring{\bb1@tempa}{\language}%
2049           {\bb1@scset\<#2name>\<#1#2name>}%
2050           {}}%
2051         \else % Old way, but defined in the new way
2052         \bb1@exp{%
2053           \\bb1@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2054           \\bb1@ifsamestring{\bb1@tempa}{\language}%
2055           {\def\<#2name>{\<#1#2name>}}%
2056           {}}%
2057         \fi%
2058       \fi
2059       \@namedef{#1#2name}{#3}%
2060       \toks@\expandafter{\bb1@captionslist}%
2061       \bb1@exp{\\in@{\<#2name>}{\the\toks@}}%
2062       \ifin@ \else

```

```

2063 \bbl@exp{\bbl@add\bbl@captionslist{\<#2name>}}%
2064 \bbl@tglobal\bbl@captionslist
2065 \fi
2066 \fi}
2067 % \def\bbl@setcaption@s#1#2#3{} % TODO. Not yet implemented (w/o 'name')

```

## 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.

```

2068 \bbl@trace{Macros related to glyphs}
2069 \def\set@low@box#1{\setbox\tw@hbox{,}\setbox\z@hbox{#1}%
2070 \dimen\z@ht\z@ \advance\dimen\z@ -\ht\tw@%
2071 \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.

```

2072 \def\save@sf@q#1{\leavevmode
2073 \begingroup
2074 \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2075 \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 `\quotedblbase`. In the OT1 encoding it is not available, therefore we make it available by lowering the normal open quote character to the baseline.

```

2076 \ProvideTextCommand{\quotedblbase}{OT1}{%
2077 \save@sf@q{\set@low@box{\textquotedblright\}}%
2078 \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.

```

2079 \ProvideTextCommandDefault{\quotedblbase}{%
2080 \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2081 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2082 \save@sf@q{\set@low@box{\textquoteright\}}%
2083 \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.

```

2084 \ProvideTextCommandDefault{\quotesinglbase}{%
2085 \UseTextSymbol{OT1}{\quotesinglbase}}

```

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

```

2086 \ProvideTextCommand{\guillemetleft}{OT1}{%
2087 \ifmmode
2088 \ll
2089 \else
2090 \save@sf@q{\nobreak
2091 \raise.2ex\hbox{\scriptscriptstyle\ll}\bbl@allowhyphens}%
2092 \fi}
2093 \ProvideTextCommand{\guillemetright}{OT1}{%
2094 \ifmmode
2095 \gg
2096 \else
2097 \save@sf@q{\nobreak

```

```

2098      \raise.2ex\hbox{$\scriptscriptstyle\gg$\bbl@allowhyphens}%
2099 \fi}
2100 \ProvideTextCommand{\guillemotleft}{OT1}{%
2101   \ifmmode
2102     \ll
2103   \else
2104     \save@sf@q{\nobreak
2105       \raise.2ex\hbox{$\scriptscriptstyle\ll$\bbl@allowhyphens}%
2106     \fi}
2107 \ProvideTextCommand{\guillemotright}{OT1}{%
2108   \ifmmode
2109     \gg
2110   \else
2111     \save@sf@q{\nobreak
2112       \raise.2ex\hbox{$\scriptscriptstyle\gg$\bbl@allowhyphens}%
2113     \fi}

```

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

```

2114 \ProvideTextCommandDefault{\guillemetleft}{%
2115   \UseTextSymbol{OT1}{\guillemetleft}}
2116 \ProvideTextCommandDefault{\guillemetright}{%
2117   \UseTextSymbol{OT1}{\guillemetright}}
2118 \ProvideTextCommandDefault{\guillemotleft}{%
2119   \UseTextSymbol{OT1}{\guillemotleft}}
2120 \ProvideTextCommandDefault{\guillemotright}{%
2121   \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2122 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2123   \ifmmode
2124     <%
2125   \else
2126     \save@sf@q{\nobreak
2127       \raise.2ex\hbox{$\scriptscriptstyle<$\bbl@allowhyphens}%
2128     \fi}
2129 \ProvideTextCommand{\guilsinglright}{OT1}{%
2130   \ifmmode
2131     >%
2132   \else
2133     \save@sf@q{\nobreak
2134       \raise.2ex\hbox{$\scriptscriptstyle>$\bbl@allowhyphens}%
2135     \fi}

```

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

```

2136 \ProvideTextCommandDefault{\guilsinglleft}{%
2137   \UseTextSymbol{OT1}{\guilsinglleft}}
2138 \ProvideTextCommandDefault{\guilsinglright}{%
2139   \UseTextSymbol{OT1}{\guilsinglright}}

```

### 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 `\IJ` fonts. Therefore we fake it for the OT1 encoding.

```

2140 \DeclareTextCommand{\ij}{OT1}{%
2141   i\kern-0.02em\bbl@allowhyphens j}
2142 \DeclareTextCommand{\IJ}{OT1}{%
2143   I\kern-0.02em\bbl@allowhyphens J}
2144 \DeclareTextCommand{\ij}{T1}{\char188}
2145 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2146 \ProvideTextCommandDefault{\ij}{%
2147   \UseTextSymbol{OT1}{\ij}}

```

```

2148 \ProvideTextCommandDefault{\IJ}{%
2149   \UseTextSymbol{OT1}{\IJ}}

```

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

```

2150 \def\crrtic@{\hrule height0.1ex width0.3em}
2151 \def\crttic@{\hrule height0.1ex width0.33em}
2152 \def\ddj@{%
2153   \setbox0\hbox{d}\dimen@=\ht0
2154   \advance\dimen@1ex
2155   \dimen@.45\dimen@
2156   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2157   \advance\dimen@ii.5ex
2158   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2159 \def\DDJ@{%
2160   \setbox0\hbox{D}\dimen@=.55\ht0
2161   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2162   \advance\dimen@ii.15ex % correction for the dash position
2163   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2164   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2165   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2166 %
2167 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2168 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2169 \ProvideTextCommandDefault{\dj}{%
2170   \UseTextSymbol{OT1}{\dj}}
2171 \ProvideTextCommandDefault{\DJ}{%
2172   \UseTextSymbol{OT1}{\DJ}}

```

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

```

2173 \DeclareTextCommand{\SS}{OT1}{SS}
2174 \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 mathmode. 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

```

2175 \ProvideTextCommandDefault{\glq}{%
2176   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
2177 \ProvideTextCommand{\grq}{T1}{%
2178   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2179 \ProvideTextCommand{\grq}{TU}{%
2180   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2181 \ProvideTextCommand{\grq}{OT1}{%
2182   \save@sf@q{\kern-.0125em
2183     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2184     \kern.07em\relax}}
2185 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}{\grq}}

```

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

\glqq The ‘german’ double quotes.

\grqq

```

2186 \ProvideTextCommandDefault{\glqq}{%
2187   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```
2188 \ProvideTextCommand{\grqq}{T1}{%
2189   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2190 \ProvideTextCommand{\grqq}{TU}{%
2191   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2192 \ProvideTextCommand{\grqq}{OT1}{%
2193   \save@sf@q{\kern-.07em
2194     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2195     \kern.07em\relax}}
2196 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq` The ‘french’ single guillemets.

```
\frq 2197 \ProvideTextCommandDefault{\flq}{%
2198   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2199 \ProvideTextCommandDefault{\frq}{%
2200   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq 2201 \ProvideTextCommandDefault{\flqq}{%
2202   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2203 \ProvideTextCommandDefault{\frqq}{%
2204   \textormath{\guillemetright}{\mbox{\guillemetright}}}
```

#### 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.

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

```
\umlautlow 2205 \def\umlauthigh{%
2206   \def\bbl@umlauta##1{\leavevmode\bgroup%
2207     \accent\csname\fontencoding dqpos\endcsname
2208     ##1\bbl@allowhyphens\egroup}%
2209   \let\bbl@umlaute\bbl@umlauta}
2210 \def\umlautlow{%
2211   \def\bbl@umlauta{\protect\lower@umlaut}}
2212 \def\umlautelow{%
2213   \def\bbl@umlaute{\protect\lower@umlaut}}
2214 \umlauthigh
```

`\lower@umlaut` The command `\lower@umlaut` is used to position the `\` closer to the letter.

We want the umlaut character lowered, nearer to the letter. To do this we need an extra *⟨dimen⟩* register.

```
2215 \expandafter\ifx\csname U@D\endcsname\relax
2216   \csname newdimen\endcsname\U@D
2217 \fi
```

The following code fools TeX’s `make_accent` procedure about the current x-height of the font to force another placement of the umlaut character. First we have to save the current x-height of the font, because we’ll change this font dimension and this is always done globally.

Then we compute the new x-height in such a way that the umlaut character is lowered to the base character. The value of `.45ex` depends on the METAFONT parameters with which the fonts were built. (Just try out, which value will look best.) If the new x-height is too low, it is not changed. Finally we call the `\accent` primitive, reset the old x-height and insert the base character in the argument.

```
2218 \def\lower@umlaut#1{%
2219   \leavevmode\bgroup
2220   \U@D 1ex%
2221   {\setbox\z@\hbox{%
2222     \char\csname\fontencoding dqpos\endcsname}%
```

```

2223 \dimen@ -.45ex\advance\dimen@ \ht\z@
2224 \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2225 \accent\csname\fontencoding dqpos\endcsname
2226 \fontdimen5\font\U@D #1%
2227 \egroup}

```

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

```

2228 \AtBeginDocument{%
2229 \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlaut{a}}%
2230 \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaut{e}}%
2231 \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaut{i}}%
2232 \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaut{\i}}%
2233 \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlaut{o}}%
2234 \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlaut{u}}%
2235 \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlaut{A}}%
2236 \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaut{E}}%
2237 \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaut{I}}%
2238 \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlaut{O}}%
2239 \DeclareTextCompositeCommand{\}{OT1}{U}{\bbl@umlaut{U}}%

```

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

```

2240 \ifx\l@english\@undefined
2241 \chardef\l@english\z@
2242 \fi
2243 % The following is used to cancel rules in ini files (see Amharic).
2244 \ifx\l@unhyphenated\@undefined
2245 \newlanguage\l@unhyphenated
2246 \fi

```

## 7.13 Layout

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

```

2247 \bbl@trace{Bidi layout}
2248 \providecommand\IfBabelLayout[3]{#3}%
2249 \newcommand\BabelPatchSection[1]{%
2250 \ifundefined{#1}{}%
2251 \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2252 \@namedef{#1}{%
2253 \ifstar{\bbl@presec@#1}%
2254 {\@dblarg{\bbl@presec@x{#1}}}}%
2255 \def\bbl@presec@x#1[#2]#3{%
2256 \bbl@exp{%
2257 \\\select@language@x{\bbl@main@language}%
2258 \\\bbl@cs{sspre@#1}%
2259 \\\bbl@cs{ss@#1}%
2260 [\\foreignlanguage{\language}{\unexpanded{#2}}}%
2261 {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2262 \\\select@language@x{\language}}%
2263 \def\bbl@presec@s#1#2{%
2264 \bbl@exp{%
2265 \\\select@language@x{\bbl@main@language}%
2266 \\\bbl@cs{sspre@#1}%
2267 \\\bbl@cs{ss@#1}*%
2268 {\\foreignlanguage{\language}{\unexpanded{#2}}}%
2269 \\\select@language@x{\language}}%
2270 \IfBabelLayout{sectioning}%
2271 {\BabelPatchSection{part}%

```



```

2272 \BabelPatchSection{chapter}%
2273 \BabelPatchSection{section}%
2274 \BabelPatchSection{subsection}%
2275 \BabelPatchSection{subsubsection}%
2276 \BabelPatchSection{paragraph}%
2277 \BabelPatchSection{subparagraph}%
2278 \def\babel@toc#1{%
2279     \select@language@x{\bbl@main@language}}{}
2280 \IfBabelLayout{captions}%
2281 {\BabelPatchSection{caption}}{}

```

## 7.14 Load engine specific macros

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

```

2282 \bbl@trace{Input engine specific macros}
2283 \ifcase\bbl@engine
2284 \input txtbabel.def
2285 \or
2286 \input luababel.def
2287 \or
2288 \input xebabel.def
2289 \fi
2290 \providecommand\babelfont{%
2291 \bbl@error
2292 {This macro is available only in LuaLaTeX and XeLaTeX.}%
2293 {Consider switching to these engines.}}
2294 \providecommand\babelprehyphenation{%
2295 \bbl@error
2296 {This macro is available only in LuaLaTeX.}%
2297 {Consider switching to that engine.}}
2298 \ifx\babelposthyphenation\@undefined
2299 \let\babelposthyphenation\babelprehyphenation
2300 \let\babelpatterns\babelprehyphenation
2301 \let\babelcharproperty\babelprehyphenation
2302 \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.

```

2303 \bbl@trace{Creating languages and reading ini files}
2304 \let\bbl@extend@ini\@gobble
2305 \newcommand\babelprovide[2][]{%
2306 \let\bbl@savelangname\language
2307 \edef\bbl@savelocaleid{\the\localeid}%
2308 % Set name and locale id
2309 \edef\language{#2}%
2310 \bbl@id@assign
2311 % Initialize keys
2312 \bbl@vforeach{captions,date,import,main,script,language,%
2313 hyphenrules,linebreaking,justification,mapfont,maparabic,%
2314 mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2315 Alph,labels,labels*,calendar,date}%
2316 {\bbl@csarg\let{KVP@##1}\@nnil}%
2317 \global\let\bbl@release@transforms\@empty
2318 \let\bbl@calendars\@empty
2319 \global\let\bbl@inidata\@empty
2320 \global\let\bbl@extend@ini\@gobble
2321 \gdef\bbl@key@list{;}%
2322 \bbl@forkv{#1}{%

```

```

2323 \in@{/{/}{##1}%
2324 \ifin@
2325 \global\let\bbl@extend@ini\bbl@extend@ini@aux
2326 \bbl@renewinikey##1\@@{##2}%
2327 \else
2328 \bbl@csarg\ifx{KVP@##1}\@nnil\else
2329 \bbl@error
2330 {Unknown key '##1' in \string\babelprovide}%
2331 {See the manual for valid keys}%
2332 \fi
2333 \bbl@csarg\def{KVP@##1}{##2}%
2334 \fi}%
2335 \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2336 \bbl@ifunset{date#2}\z{\bbl@ifunset{\bbl@llevel@#2}\@ne\tw@}%
2337 % == init ==
2338 \ifx\bbl@screset\@undefined
2339 \bbl@ldfinit
2340 \fi
2341 % == date (as option) ==
2342 % \ifx\bbl@KVP@date\@nnil\else
2343 % \fi
2344 % ==
2345 \let\bbl@lbkflag\relax % \@empty = do setup linebreak
2346 \ifcase\bbl@howloaded
2347 \let\bbl@lbkflag\@empty % new
2348 \else
2349 \ifx\bbl@KVP@hyphenrules\@nnil\else
2350 \let\bbl@lbkflag\@empty
2351 \fi
2352 \ifx\bbl@KVP@import\@nnil\else
2353 \let\bbl@lbkflag\@empty
2354 \fi
2355 \fi
2356 % == import, captions ==
2357 \ifx\bbl@KVP@import\@nnil\else
2358 \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
2359 {\ifx\bbl@initoload\relax
2360 \begingroup
2361 \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2362 \bbl@input@texini{#2}%
2363 \endgroup
2364 \else
2365 \xdef\bbl@KVP@import{\bbl@initoload}%
2366 \fi}%
2367 {}%
2368 \let\bbl@KVP@date\@empty
2369 \fi
2370 \ifx\bbl@KVP@captions\@nnil
2371 \let\bbl@KVP@captions\bbl@KVP@import
2372 \fi
2373 % ==
2374 \ifx\bbl@KVP@transforms\@nnil\else
2375 \bbl@replace\bbl@KVP@transforms{ }{,}%
2376 \fi
2377 % == Load ini ==
2378 \ifcase\bbl@howloaded
2379 \bbl@provide@new{#2}%
2380 \else
2381 \bbl@ifblank{#1}%
2382 {}% With \bbl@load@basic below
2383 {\bbl@provide@renew{#2}}%
2384 \fi
2385 % Post tasks

```

```

2386 % -----
2387 % == subsequent calls after the first provide for a locale ==
2388 \ifx\bbbl@inidata\@empty\else
2389   \bbbl@extend@ini{#2}%
2390 \fi
2391 % == ensure captions ==
2392 \ifx\bbbl@KVP@captions\@nnil\else
2393   \bbbl@ifunset{bbbl@extracaps@#2}%
2394     {\bbbl@exp{\bbabelensure[exclude=\\today]{#2}}}%
2395     {\bbbl@exp{\bbabelensure[exclude=\\today,
2396       include=\[bbbl@extracaps@#2]]{#2}}}%
2397   \bbbl@ifunset{bbbl@ensure@language}%
2398     {\bbbl@exp{%
2399       \\DeclareRobustCommand\<bbbl@ensure@language>[1]{%
2400         \\foreignlanguage{language}%
2401         {###1}}}%
2402     }%
2403   \bbbl@exp{%
2404     \\bbbl@tglobal\<bbbl@ensure@language>%
2405     \\bbbl@tglobal\<bbbl@ensure@language\space>%
2406   \fi
2407 % ==
2408 % At this point all parameters are defined if 'import'. Now we
2409 % execute some code depending on them. But what about if nothing was
2410 % imported? We just set the basic parameters, but still loading the
2411 % whole ini file.
2412 \bbbl@load@basic{#2}%
2413 % == script, language ==
2414 % Override the values from ini or defines them
2415 \ifx\bbbl@KVP@script\@nnil\else
2416   \bbbl@csarg\edef{sname@#2}{\bbbl@KVP@script}%
2417 \fi
2418 \ifx\bbbl@KVP@language\@nnil\else
2419   \bbbl@csarg\edef{lname@#2}{\bbbl@KVP@language}%
2420 \fi
2421 \ifcase\bbbl@engine\or
2422   \bbbl@ifunset{bbbl@chrng@language}{}%
2423   {\directlua{
2424     Babel.set_chranges_b('\bbbl@cl{sbcpr}', '\bbbl@cl{chrng}') }}%
2425 \fi
2426 % == onchar ==
2427 \ifx\bbbl@KVP@onchar\@nnil\else
2428   \bbbl@luahyphenate
2429   \bbbl@exp{%
2430     \\AddToHook{env/document/before}{\select@language{#2}}}%
2431   \directlua{
2432     if Babel.locale_mapped == nil then
2433       Babel.locale_mapped = true
2434       Babel.linebreaking.add_before(Babel.locale_map)
2435       Babel.loc_to_scr = {}
2436       Babel.chr_to_loc = Babel.chr_to_loc or {}
2437     end
2438     Babel.locale_props[\the\localeid].letters = false
2439   }%
2440   \bbbl@xin@{ letters }{ \bbbl@KVP@onchar\space}%
2441   \ifin@
2442     \directlua{
2443       Babel.locale_props[\the\localeid].letters = true
2444     }%
2445   \fi
2446   \bbbl@xin@{ ids }{ \bbbl@KVP@onchar\space}%
2447   \ifin@
2448     \ifx\bbbl@starthyphens\undefined % Needed if no explicit selection

```

```

2449     \AddBabelHook{babel-onchar}{beforestart}{\bb@starthyphens}}%
2450 \fi
2451 \bb@exp{\bb@add{\bb@starthyphens
2452   {\bb@patterns@lua{\language}}}%
2453 % TODO - error/warning if no script
2454 \directlua{
2455   if Babel.script_blocks['\bb@cl{sbc}'] then
2456     Babel.loc_to_scr[\the\localeid] =
2457       Babel.script_blocks['\bb@cl{sbc}']
2458     Babel.locale_props[\the\localeid].lc = \the\localeid\space
2459     Babel.locale_props[\the\localeid].lg = \the\nameuse{1@\language}\space
2460   end
2461 }%
2462 \fi
2463 \bb@xin@{ fonts }{ \bb@KVP@onchar\space}%
2464 \ifin@
2465   \bb@ifunset{bb@lsys@\language}{\bb@provide@lsys{\language}}{}%
2466   \bb@ifunset{bb@wdir@\language}{\bb@provide@dirs{\language}}{}%
2467   \directlua{
2468     if Babel.script_blocks['\bb@cl{sbc}'] then
2469       Babel.loc_to_scr[\the\localeid] =
2470         Babel.script_blocks['\bb@cl{sbc}']
2471     end}%
2472   \ifx\bb@mapselect\undefined % TODO. almost the same as mapfont
2473     \AtBeginDocument{%
2474       \bb@patchfont{\bb@mapselect}}%
2475     {\selectfont}}%
2476   \def\bb@mapselect{%
2477     \let\bb@mapselect\relax
2478     \edef\bb@prefontid{\fontid\font}}%
2479   \def\bb@mapdir##1{%
2480     {\def\language{##1}%
2481     \let\bb@ifrestoring\@firstoftwo % To avoid font warning
2482     \bb@switchfont
2483     \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2484       \directlua{
2485         Babel.locale_props[\the\csname bb@id@##1\endcsname]%
2486           ['/\bb@prefontid'] = \fontid\font\space}%
2487       \fi}}%
2488   \fi
2489   \bb@exp{\bb@add{\bb@mapselect{\bb@mapdir{\language}}}}%
2490 \fi
2491 % TODO - catch non-valid values
2492 \fi
2493 % == mapfont ==
2494 % For bidi texts, to switch the font based on direction
2495 \ifx\bb@KVP@mapfont\@nnil\else
2496   \bb@ifsamestring{\bb@KVP@mapfont}{direction}}{}%
2497   {\bb@error{Option '\bb@KVP@mapfont' unknown for\%
2498     mapfont. Use 'direction'.%
2499     {See the manual for details.}}}%
2500   \bb@ifunset{bb@lsys@\language}{\bb@provide@lsys{\language}}{}%
2501   \bb@ifunset{bb@wdir@\language}{\bb@provide@dirs{\language}}{}%
2502   \ifx\bb@mapselect\undefined % TODO. See onchar.
2503     \AtBeginDocument{%
2504       \bb@patchfont{\bb@mapselect}}%
2505     {\selectfont}}%
2506   \def\bb@mapselect{%
2507     \let\bb@mapselect\relax
2508     \edef\bb@prefontid{\fontid\font}}%
2509   \def\bb@mapdir##1{%
2510     {\def\language{##1}%
2511     \let\bb@ifrestoring\@firstoftwo % avoid font warning

```

```

2512         \bbl@switchfont
2513         \directlua{Babel.fontmap
2514             [\the\csname bbl@wdir@##1\endcsname]%
2515             [\bbl@prefontid]=\fontid\font}}}%
2516     \fi
2517     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language\language}}}%
2518 \fi
2519 % == Line breaking: intraspace, intrapenalty ==
2520 % For CJK, East Asian, Southeast Asian, if interspace in ini
2521 \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2522     \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2523 \fi
2524 \bbl@provide@intraspace
2525 % == Line breaking: CJK quotes ==
2526 \ifcase\bbl@engine\or
2527     \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
2528 \ifin@
2529     \bbl@ifunset{bbl@quote@\language\language}{}%
2530     {\directlua{
2531         Babel.locale_props[\the\localeid].cjk_quotes = {}
2532         local cs = 'op'
2533         for c in string.utfvalues(
2534             [[\csname bbl@quote@\language\language\endcsname]]) do
2535             if Babel.cjk_characters[c].c == 'qu' then
2536                 Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2537             end
2538             cs = (cs == 'op') and 'cl' or 'op'
2539         end
2540     }}%
2541 \fi
2542 \fi
2543 % == Line breaking: justification ==
2544 \ifx\bbl@KVP@justification\@nnil\else
2545     \let\bbl@KVP@linebreaking\bbl@KVP@justification
2546 \fi
2547 \ifx\bbl@KVP@linebreaking\@nnil\else
2548     \bbl@xin@{,\bbl@KVP@linebreaking,%
2549         {,elongated,kashida,cjk,padding,unhyphenated,%
2550         \ifin@
2551             \bbl@csarg\xdef
2552             {\lnbrk@\language\language}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2553 \fi
2554 \fi
2555 \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
2556 \ifin@\else\bbl@xin@{/k}{/\bbl@cl{lnbrk}}\fi
2557 \ifin@\bbl@arabicjust\fi
2558 \bbl@xin@{/p}{/\bbl@cl{lnbrk}}%
2559 \ifin@\AtBeginDocument{\@nameuse{bbl@tibetanjust}}\fi
2560 % == Line breaking: hyphenate.other.(locale|script) ==
2561 \ifx\bbl@lbfkflag\@empty
2562     \bbl@ifunset{bbl@hyotl@\language\language}{}%
2563     {\bbl@csarg\bbl@replace{hyotl@\language\language}{ }{,}%
2564     \bbl@startcommands*\language\language}%
2565     \bbl@csarg\bbl@foreach{hyotl@\language\language}{%
2566         \ifcase\bbl@engine
2567             \ifnum##1<257
2568                 \SetHyphenMap{\BabelLower{##1}{##1}}%
2569             \fi
2570         \else
2571             \SetHyphenMap{\BabelLower{##1}{##1}}%
2572         \fi}%
2573     \bbl@endcommands}%
2574 \bbl@ifunset{bbl@hyots@\language\language}{}%

```

```

2575     {\bbl@csarg\bbl@replace{hyots@language}{ }{ },}%
2576     \bbl@csarg\bbl@foreach{hyots@language}{%
2577         \ifcase\bbl@engine
2578             \ifnum##1<257
2579                 \global\lccode##1=##1\relax
2580             \fi
2581         \else
2582             \global\lccode##1=##1\relax
2583         \fi}}%
2584 \fi
2585 % == Counters: maparabic ==
2586 % Native digits, if provided in ini (TeX level, xe and lua)
2587 \ifcase\bbl@engine\else
2588     \bbl@ifunset{\bbl@dgnat@language}{}%
2589     {\expandafter\ifx\csname bbl@dgnat@language\endcsname\@empty\else
2590         \expandafter\expandafter\expandafter
2591         \bbl@setdigits\csname bbl@dgnat@language\endcsname
2592         \ifx\bbl@KVP@maparabic\@nnil\else
2593             \ifx\bbl@latinarabic\@undefined
2594                 \expandafter\let\expandafter\@arabic
2595                 \csname bbl@counter@language\endcsname
2596             \else % ie, if layout=counters, which redefines \@arabic
2597                 \expandafter\let\expandafter\bbl@latinarabic
2598                 \csname bbl@counter@language\endcsname
2599             \fi
2600         \fi
2601     \fi}%
2602 \fi
2603 % == Counters: mapdigits ==
2604 % Native digits (lua level).
2605 \ifodd\bbl@engine
2606     \ifx\bbl@KVP@mapdigits\@nnil\else
2607         \bbl@ifunset{\bbl@dgnat@language}{}%
2608         {\RequirePackage{luatexbase}%
2609         \bbl@activate@preotf
2610         \directlua{
2611             Babel = Babel or {} %%% -> presets in luababel
2612             Babel.digits_mapped = true
2613             Babel.digits = Babel.digits or {}
2614             Babel.digits[\the\localeid] =
2615                 table.pack(string.utfvalue('\bbl@cl{dgnat}'))
2616             if not Babel.numbers then
2617                 function Babel.numbers(head)
2618                     local LOCALE = Babel.attr_locale
2619                     local GLYPH = node.id'glyph'
2620                     local inmath = false
2621                     for item in node.traverse(head) do
2622                         if not inmath and item.id == GLYPH then
2623                             local temp = node.get_attribute(item, LOCALE)
2624                             if Babel.digits[temp] then
2625                                 local chr = item.char
2626                                 if chr > 47 and chr < 58 then
2627                                     item.char = Babel.digits[temp][chr-47]
2628                                 end
2629                             end
2630                         elseif item.id == node.id'math' then
2631                             inmath = (item.subtype == 0)
2632                         end
2633                     end
2634                     return head
2635                 end
2636             end
2637         }}%

```

```

2638 \fi
2639 \fi
2640 % == Counters: alph, Alph ==
2641 % What if extras<lang> contains a \babel@save\@alph? It won't be
2642 % restored correctly when exiting the language, so we ignore
2643 % this change with the \bbl@alph@saved trick.
2644 \ifx\bbl@KVP@alph\@nnil\else
2645 \bbl@extras@wrap{\bbl@alph@saved}%
2646 {\let\bbl@alph@saved\@alph}%
2647 {\let\@alph\bbl@alph@saved
2648 \babel@save\@alph}%
2649 \bbl@exp{%
2650 \bbl@add\<extras\languagename>{%
2651 \let\@alph\<bbl@cntr@\bbl@KVP@alph @\languagename>}}%
2652 \fi
2653 \ifx\bbl@KVP@Alph\@nnil\else
2654 \bbl@extras@wrap{\bbl@Alph@saved}%
2655 {\let\bbl@Alph@saved\@Alph}%
2656 {\let\@Alph\bbl@Alph@saved
2657 \babel@save\@Alph}%
2658 \bbl@exp{%
2659 \bbl@add\<extras\languagename>{%
2660 \let\@Alph\<bbl@cntr@\bbl@KVP@Alph @\languagename>}}%
2661 \fi
2662 % == Calendars ==
2663 \ifx\bbl@KVP@calendar\@nnil
2664 \edef\bbl@KVP@calendar{\bbl@c1{calpr}}%
2665 \fi
2666 \def\bbl@tempe##1 ##2\@{% % Get first calendar
2667 \def\bbl@tempa{##1}}%
2668 \bbl@exp{\bbl@tempe\bbl@KVP@calendar\space\@}%
2669 \def\bbl@tempe##1.##2.##3\@{%
2670 \def\bbl@tempc{##1}%
2671 \def\bbl@tempb{##2}}%
2672 \expandafter\bbl@tempe\bbl@tempa..\@
2673 \bbl@csarg\edef{calpr@\languagename}{%
2674 \ifx\bbl@tempc\@empty\else
2675 calendar=\bbl@tempc
2676 \fi
2677 \ifx\bbl@tempb\@empty\else
2678 ,variant=\bbl@tempb
2679 \fi}%
2680 % == require.babel in ini ==
2681 % To load or reload the babel-*.tex, if require.babel in ini
2682 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
2683 \bbl@ifunset{\bbl@rqtex@\languagename}{}%
2684 {\expandafter\ifx\csname\bbl@rqtex@\languagename\endcsname\@empty\else
2685 \let\BabelBeforeIni\@gobbles
2686 \chardef\atcatcode=\catcode\@
2687 \catcode\@=11\relax
2688 \bbl@input@texini{\bbl@cs{rqtex@\languagename}}%
2689 \catcode\@=\atcatcode
2690 \let\atcatcode\relax
2691 \global\bbl@csarg\let{rqtex@\languagename}\relax
2692 \fi}%
2693 \bbl@foreach\bbl@calendars{%
2694 \bbl@ifunset{\bbl@ca##1}{%
2695 \chardef\atcatcode=\catcode\@
2696 \catcode\@=11\relax
2697 \InputIfFileExists{babel-ca-##1.tex}{}}%
2698 \catcode\@=\atcatcode
2699 \let\atcatcode\relax}%
2700 {}}%

```

```

2701 \fi
2702 % == frenchspacing ==
2703 \ifcase\bb1@howloaded\in@true\else\in@false\fi
2704 \ifin@\else\bb1@xin@{typography/frenchspacing}{\bb1@key@list}\fi
2705 \ifin@
2706 \bb1@extras@wrap{\bb1@pre@fs}%
2707 {\bb1@pre@fs}%
2708 {\bb1@post@fs}%
2709 \fi
2710 % == transforms ==
2711 \ifodd\bb1@engine
2712 \ifx\bb1@KVP@transforms\@nnil\else
2713 \def\bb1@elt##1##2##3{%
2714 \in@{$transforms.}{$##1}%
2715 \ifin@
2716 \def\bb1@tempa{##1}%
2717 \bb1@replace\bb1@tempa{transforms.}{}%
2718 \bb1@carg\bb1@transforms{babel\bb1@tempa}{##2}{##3}%
2719 \fi}%
2720 \csname bbl@inidata@\language\endcsname
2721 \bb1@release@transforms\relax % \relax closes the last item.
2722 \fi
2723 \fi
2724 % == main ==
2725 \ifx\bb1@KVP@main\@nnil % Restore only if not 'main'
2726 \let\language\bb1@savelangname
2727 \chardef\localeid\bb1@savelocaleid\relax
2728 \fi}

```

Depending on whether or not the language exists (based on \date<language>), we define two macros. Remember \bb1@startcommands opens a group.

```

2729 \def\bb1@provide@new#1{%
2730 \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2731 \@namedef{extras#1}{}%
2732 \@namedef{noextras#1}{}%
2733 \bb1@startcommands*{#1}{captions}%
2734 \ifx\bb1@KVP@captions\@nnil % and also if import, implicit
2735 \def\bb1@tempb##1{% elt for \bb1@captionslist
2736 \ifx##1\@empty\else
2737 \bb1@exp{%
2738 \SetString\##1{%
2739 \bb1@nocaption{\bb1@stripslash##1}{#1\bb1@stripslash##1}}}%
2740 \expandafter\bb1@tempb
2741 \fi}%
2742 \expandafter\bb1@tempb\bb1@captionslist\@empty
2743 \else
2744 \ifx\bb1@initoload\relax
2745 \bb1@read@ini{\bb1@KVP@captions}2% % Here letters cat = 11
2746 \else
2747 \bb1@read@ini{\bb1@initoload}2% % Same
2748 \fi
2749 \fi
2750 \StartBabelCommands*{#1}{date}%
2751 \ifx\bb1@KVP@date\@nnil
2752 \bb1@exp{%
2753 \SetString\today{\bb1@nocaption{today}{#1today}}}%
2754 \else
2755 \bb1@savetoday
2756 \bb1@savedate
2757 \fi
2758 \bb1@endcommands
2759 \bb1@load@basic{#1}%
2760 % == hyphenmins == (only if new)

```



```

2761 \bbl@exp{%
2762 \gdef\<#1hyphenmins>{%
2763 {\bbl@ifunset{\bbl@ifthm#1}{2}{\bbl@cs{lftm#1}}}%
2764 {\bbl@ifunset{\bbl@rgthm#1}{3}{\bbl@cs{rgthm#1}}}%
2765 % == hyphenrules (also in renew) ==
2766 \bbl@provide@hyphens{#1}%
2767 \ifx\bbl@KVP@main\@nnil\else
2768 \expandafter\main@language\expandafter{#1}%
2769 \fi}
2770 %
2771 \def\bbl@provide@renew#1{%
2772 \ifx\bbl@KVP@captions\@nnil\else
2773 \StartBabelCommands*{#1}{captions}%
2774 \bbl@read@ini{\bbl@KVP@captions}2% % Here all letters cat = 11
2775 \EndBabelCommands
2776 \fi
2777 \ifx\bbl@KVP@date\@nnil\else
2778 \StartBabelCommands*{#1}{date}%
2779 \bbl@savetoday
2780 \bbl@savestate
2781 \EndBabelCommands
2782 \fi
2783 % == hyphenrules (also in new) ==
2784 \ifx\bbl@lbkflag\@empty
2785 \bbl@provide@hyphens{#1}%
2786 \fi}

```

Load the basic parameters (ids, typography, counters, and a few more), while captions and dates are left out. But it may happen some data has been loaded before automatically, so we first discard the saved values. (TODO. But preserving previous values would be useful.)

```

2787 \def\bbl@load@basic#1{%
2788 \ifcase\bbl@howloaded\or\or
2789 \ifcase\csname bbl@llevel\language\endcsname
2790 \bbl@csarg\let{\lname@language}\relax
2791 \fi
2792 \fi
2793 \bbl@ifunset{\bbl@lname@#1}%
2794 {\def\BabelBeforeIni##1##2{%
2795 \begingroup
2796 \let\bbl@ini@captions@aux\@gobbletwo
2797 \def\bbl@inidate ####1.###2.###3.###4\relax ####5###6}%
2798 \bbl@read@ini{##1}1%
2799 \ifx\bbl@initoload\relax\endinput\fi
2800 \endgroup}%
2801 \begingroup % boxed, to avoid extra spaces:
2802 \ifx\bbl@initoload\relax
2803 \bbl@input@texini{#1}%
2804 \else
2805 \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
2806 \fi
2807 \endgroup}%
2808 {}

```

The hyphenrules option is handled with an auxiliary macro.

```

2809 \def\bbl@provide@hyphens#1{%
2810 \let\bbl@tempa\relax
2811 \ifx\bbl@KVP@hyphenrules\@nnil\else
2812 \bbl@replace\bbl@KVP@hyphenrules{ },}%
2813 \bbl@foreach\bbl@KVP@hyphenrules{%
2814 \ifx\bbl@tempa\relax % if not yet found
2815 \bbl@ifsamestring{##1}{+}%
2816 {\bbl@exp{\addlanguage\<l@##1>}}%
2817 }%
2818 \bbl@ifunset{l@##1}%

```

```

2819         {}%
2820         {\bbl@exp{\let\bbl@tempa<l@##1>}}%
2821     \fi}%
2822 \fi
2823 \ifx\bbl@tempa\relax %           if no opt or no language in opt found
2824     \ifx\bbl@KVP@import\@nnil
2825         \ifx\bbl@initoload\relax\else
2826             \bbl@exp{%           and hyphenrules is not empty
2827                 \\bbl@ifblank{\bbl@cs{hyphr@#1}}%
2828                 {}%
2829                 {\let\\bbl@tempa<l@>\bbl@cl{hyphr}>}}%
2830     \fi
2831 \else % if importing
2832     \bbl@exp{%           and hyphenrules is not empty
2833         \\bbl@ifblank{\bbl@cs{hyphr@#1}}%
2834         {}%
2835         {\let\\bbl@tempa<l@>\bbl@cl{hyphr}>}}%
2836 \fi
2837 \fi
2838 \bbl@ifunset{bbl@tempa}%       ie, relax or undefined
2839 {\bbl@ifunset{l@#1}%           no hyphenrules found - fallback
2840     {\bbl@exp{\\adddialect<l@#1>\language}}%
2841     {}}%                       so, l@<lang> is ok - nothing to do
2842 {\bbl@exp{\\adddialect<l@#1>\bbl@tempa}}}% found in opt list or ini

```

The reader of babel-...tex files. We reset temporarily some catcodes.

```

2843 \def\bbl@input@texini#1{%
2844     \bbl@bsphack
2845     \bbl@exp{%
2846         \catcode`\\%=14 \catcode`\\=0
2847         \catcode`\\={1 \catcode`\\}=2
2848         \lowercase{\\InputIfFileExists{babel-#1.tex}{}}}%
2849         \catcode`\\%=\the\catcode`\% \relax
2850         \catcode`\\=\the\catcode`\% \relax
2851         \catcode`\\={\the\catcode`\% \relax
2852         \catcode`\\=\the\catcode`\% \relax}%
2853     \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.

```

2854 \def\bbl@inline#1\bbl@inline{%
2855     \ifnextchar[\bbl@iniset{\ifnextchar\bbl@iniskip\bbl@inistore}#1\@@% ]
2856 \def\bbl@iniset[#1]#2\@@{\def\bbl@section{#1}}
2857 \def\bbl@iniskip#1\@@{}%       if starts with ;
2858 \def\bbl@inistore#1=#2\@@{}%   full (default)
2859 \bbl@trim@def\bbl@tempa{#1}%
2860 \bbl@trim\toks@{#2}%
2861 \bbl@xin@;\bbl@section/\bbl@tempa;{\bbl@key@list}%
2862 \ifin@ \else
2863     \bbl@xin@{,identification/include.}%
2864     {\bbl@section/\bbl@tempa}%
2865 \ifin@ \edef\bbl@required@inis{\the\toks@} \fi
2866 \bbl@exp{%
2867     \\g@addto@macro\\bbl@inidata{%
2868         \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2869 \fi}
2870 \def\bbl@inistore@min#1=#2\@@{}% minimal (maybe set in \bbl@read@ini)
2871 \bbl@trim@def\bbl@tempa{#1}%
2872 \bbl@trim\toks@{#2}%
2873 \bbl@xin@{.identification.}{.\bbl@section.}%
2874 \ifin@
2875     \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2876         \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%

```

2877 \fi}

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

```

2878 \def\bbl@loop@ini{%
2879   \loop
2880     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2881     \endlinechar\m@ne
2882     \read\bbl@readstream to \bbl@line
2883     \endlinechar`^^M
2884     \ifx\bbl@line\@empty\else
2885       \expandafter\bbl@iniline\bbl@line\bbl@iniline
2886     \fi
2887   \repeat}
2888 \ifx\bbl@readstream\@undefined
2889   \csname newread\endcsname\bbl@readstream
2890 \fi
2891 \def\bbl@read@ini#1#2{%
2892   \global\let\bbl@extend@ini\@gobble
2893   \openin\bbl@readstream=babel-#1.ini
2894   \ifeof\bbl@readstream
2895     \bbl@error
2896     {There is no ini file for the requested language\\%
2897      (#1: \language). Perhaps you misspelled it or your\\%
2898      installation is not complete.}%
2899     {Fix the name or reinstall babel.}%
2900   \else
2901     % == Store ini data in \bbl@inidata ==
2902     \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2903     \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2904     \bbl@info{Importing
2905               \ifcase#2font and identification \or basic \fi
2906               data for \language\\%
2907               from babel-#1.ini. Reported}%
2908     \ifnum#2=\z@
2909       \global\let\bbl@inidata\@empty
2910       \let\bbl@inistore\bbl@inistore@min % Remember it's local
2911     \fi
2912     \def\bbl@section{identification}%
2913     \let\bbl@required@inis\@empty
2914     \bbl@exp{\bbl@inistore tag.ini=#1\\@@}%
2915     \bbl@inistore load.level=#2\\@@
2916     \bbl@loop@ini
2917     \ifx\bbl@required@inis\@empty\else
2918       \bbl@replace\bbl@required@inis{ }{,}%
2919       \bbl@foreach\bbl@required@inis{%
2920         \openin\bbl@readstream=##1.ini
2921         \bbl@loop@ini}%
2922     \fi
2923     % == Process stored data ==
2924     \bbl@csarg\xdef{lini@\language}{#1}%
2925     \bbl@read@ini@aux
2926     % == 'Export' data ==
2927     \bbl@ini@exports{#2}%
2928     \global\bbl@csarg\let{inidata@\language}\bbl@inidata
2929     \global\let\bbl@inidata\@empty
2930     \bbl@exp{\bbl@add@list\bbl@ini@loaded{\language}}%
2931     \bbl@tglobal\bbl@ini@loaded
2932   \fi}

```

```

2933 \def\bbl@read@ini@aux{%
2934   \let\bbl@savestrings\@empty
2935   \let\bbl@savetoday\@empty
2936   \let\bbl@savestate\@empty
2937   \def\bbl@elt##1##2##3{%
2938     \def\bbl@section{##1}%
2939     \in@{=date.}{=##1}% Find a better place
2940     \ifin@
2941       \bbl@ifunset{bbl@inikv@##1}%
2942       {\bbl@ini@calendar{##1}}%
2943       {}%
2944     \fi
2945     \in@{=identification/extension.}{=##1/##2}%
2946     \ifin@
2947       \bbl@ini@extension{##2}%
2948     \fi
2949     \bbl@ifunset{bbl@inikv@##1}{}%
2950     {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2951   \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.

```

2952 \def\bbl@extend@ini@aux#1{%
2953   \bbl@startcommands*{#1}{captions}%
2954   % Activate captions/... and modify exports
2955   \bbl@csarg\def{inikv@captions.licr}##1##2{%
2956     \setlocalecaption{#1}{##1}{##2}}%
2957   \def\bbl@inikv@captions##1##2{%
2958     \bbl@ini@captions@aux{##1}{##2}}%
2959   \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2960   \def\bbl@exportkey##1##2##3{%
2961     \bbl@ifunset{bbl@kv@##2}{}%
2962     {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2963       \bbl@exp{\global\let\<bbl@##1@language\>\<bbl@kv@##2>}}%
2964     \fi}}%
2965   % As with \bbl@read@ini, but with some changes
2966   \bbl@read@ini@aux
2967   \bbl@ini@exports\tw@
2968   % Update inidata@lang by pretending the ini is read.
2969   \def\bbl@elt##1##2##3{%
2970     \def\bbl@section{##1}%
2971     \bbl@inline##2=##3\bbl@inline}%
2972     \csname bbl@inidata@#1\endcsname
2973     \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2974   \StartBabelCommands*{#1}{date}% And from the import stuff
2975   \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2976   \bbl@savetoday
2977   \bbl@savestate
2978   \bbl@endcommands}

```

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

```

2979 \def\bbl@ini@calendar#1{%
2980   \lowercase{\def\bbl@tempa{=##1=}}%
2981   \bbl@replace\bbl@tempa{=date.gregorian}{}%
2982   \bbl@replace\bbl@tempa{=date.}{}%
2983   \in@{.licr=}{#1=}%
2984   \ifin@
2985     \ifcase\bbl@engine
2986       \bbl@replace\bbl@tempa{.licr=}{}%
2987     \else
2988       \let\bbl@tempa\relax
2989     \fi
2990   \fi
2991   \ifx\bbl@tempa\relax\else

```

```

2992 \bbl@replace\bbl@tempa{=}{}%
2993 \ifx\bbl@tempa\@empty\else
2994 \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2995 \fi
2996 \bbl@exp{%
2997 \def<\bbl@inikv@#1>####1####2{%
2998 \\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2999 \fi}

```

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

```

3000 \def\bbl@renewinikey#1/#2\@#3{%
3001 \edef\bbl@tempa{\zap@space #1 \@empty}% section
3002 \edef\bbl@tempb{\zap@space #2 \@empty}% key
3003 \bbl@trim\toks@{#3}% value
3004 \bbl@exp{%
3005 \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
3006 \\g@addto@macro\\bbl@inidata{%
3007 \\bbl@elt{\bbl@tempa}{\bbl@tempb}{\the\toks@}}}%

```

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

```

3008 \def\bbl@exportkey#1#2#3{%
3009 \bbl@ifunset{\bbl@kv@#2}%
3010 {\bbl@csarg\gdef{#1@\language\name}{#3}}%
3011 {\expandafter\ifx\csname \bbl@kv@#2\endcsname\@empty
3012 \bbl@csarg\gdef{#1@\language\name}{#3}}%
3013 \else
3014 \bbl@exp{\global\let<\bbl@#1@\language\name>\<\bbl@kv@#2>}%
3015 \fi}}

```

Key-value pairs are treated differently depending on the section in the ini file. The following macros are the readers for identification and typography. Note \bbl@ini@exports is called always (via \bbl@inisec), while \bbl@after@ini must be called explicitly after \bbl@read@ini if necessary.

```

3016 \def\bbl@iniwarning#1{%
3017 \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
3018 {\bbl@warning{%
3019 From babel-\bbl@cs{lini@\language\name}.ini:\%
3020 \bbl@cs{@kv@identification.warning#1}\%
3021 Reported }}}
3022 %
3023 \let\bbl@release@transforms\@empty

```

BCP 47 extensions are separated by a single letter (eg, latin-x-medieval. The following macro handles this special case to create correctly the correspondig info.

```

3024 \def\bbl@ini@extension#1{%
3025 \def\bbl@tempa{#1}%
3026 \bbl@replace\bbl@tempa{extension.}{}%
3027 \bbl@replace\bbl@tempa{.tag.bcp47}{}%
3028 \bbl@ifunset{\bbl@info@#1}%
3029 {\bbl@csarg\xdef{info@#1}{ext/\bbl@tempa}%
3030 \bbl@exp{%
3031 \\g@addto@macro\\bbl@moreinfo{%
3032 \\bbl@exportkey{ext/\bbl@tempa}{identification.#1}{}}}%
3033 {}%
3034 \let\bbl@moreinfo\@empty
3035 %
3036 \def\bbl@ini@exports#1{%
3037 % Identification always exported
3038 \bbl@iniwarning{}%
3039 \ifcase\bbl@engine
3040 \bbl@iniwarning{.pdf\latex}%

```

```

3041 \or
3042 \bbl@iniwarning{.lualatex}%
3043 \or
3044 \bbl@iniwarning{.xelatex}%
3045 \fi%
3046 \bbl@exportkey{llevel}{identification.load.level}{}%
3047 \bbl@exportkey{elname}{identification.name.english}{}%
3048 \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
3049 {\csname bbl@elname@languagename\endcsname}}%
3050 \bbl@exportkey{tbcpr}{identification.tag.bcp47}{}%
3051 \bbl@exportkey{lbcpr}{identification.language.tag.bcp47}{}%
3052 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3053 \bbl@exportkey{esname}{identification.script.name}{}%
3054 \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
3055 {\csname bbl@esname@languagename\endcsname}}%
3056 \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
3057 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3058 \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
3059 \bbl@exportkey{vbcpr}{identification.variant.tag.bcp47}{}%
3060 \bbl@moreinfo
3061 % Also maps bcp47 -> languagename
3062 \ifbbl@bcptoname
3063 \bbl@csarg\xdef{bcp@map@bbl@cl{tbcpr}}{languagename}%
3064 \fi
3065 % Conditional
3066 \ifnum#1>z@ % 0 = only info, 1, 2 = basic, (re)new
3067 \bbl@exportkey{calpr}{date.calendar.preferred}{}%
3068 \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3069 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3070 \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
3071 \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3072 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3073 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3074 \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3075 \bbl@exportkey{intsp}{typography.intraspaces}{u}%
3076 \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3077 \bbl@exportkey{chrng}{characters.ranges}{}%
3078 \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
3079 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3080 \ifnum#1=\tw@ % only (re)new
3081 \bbl@exportkey{rqtex}{identification.require.babel}{}%
3082 \bbl@tglobal\bbl@savetoday
3083 \bbl@tglobal\bbl@savestate
3084 \bbl@savestrings
3085 \fi
3086 \fi}

```

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

```

3087 \def\bbl@inikv#1#2{%      key=value
3088 \toks@{#2}%              This hides #'s from ini values
3089 \bbl@csarg\edef{@kv@bbl@section.#1}{\the\toks@}}

```

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

```

3090 \let\bbl@inikv@identification\bbl@inikv
3091 \let\bbl@inikv@date\bbl@inikv
3092 \let\bbl@inikv@typography\bbl@inikv
3093 \let\bbl@inikv@characters\bbl@inikv
3094 \let\bbl@inikv@numbers\bbl@inikv

```

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

```

3095 \def\bbl@inikv@counters#1#2{%
3096 \bbl@ifsamestring{#1}{digits}%

```

```

3097     {\bbl@error{The counter name 'digits' is reserved for mapping\\%
3098             decimal digits}%
3099             {Use another name.}}%
3100     {}%
3101     \def\bbl@tempc{#1}%
3102     \bbl@trim@def{\bbl@tempb*}{#2}%
3103     \in@{.1$}{#1$}%
3104     \ifin@
3105         \bbl@replace\bbl@tempc{.1}{}%
3106         \bbl@csarg\protected@xdef{cnt@{\bbl@tempc @\language}\bbl@tempc}%
3107         \noexpand\bbl@alphanumeric{\bbl@tempc}%
3108     \fi
3109     \in@{.F.}{#1}%
3110     \ifin@\else\in@{.S.}{#1}\fi
3111     \ifin@
3112         \bbl@csarg\protected@xdef{cnt@#1@\language}\bbl@tempb*}%
3113     \else
3114         \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3115         \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3116         \bbl@csarg{\global\expandafter\let}{cnt@#1@\language}\bbl@tempa
3117     \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.

```

3118 \ifcase\bbl@engine
3119     \bbl@csarg\def{inikv@captions.licr}#1#2{%
3120         \bbl@ini@captions@aux{#1}{#2}}
3121 \else
3122     \def\bbl@inikv@captions#1#2{%
3123         \bbl@ini@captions@aux{#1}{#2}}
3124 \fi

```

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

```

3125 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3126     \bbl@replace\bbl@tempa{.template}{}%
3127     \def\bbl@toreplace{#1}{}%
3128     \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
3129     \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3130     \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3131     \bbl@replace\bbl@toreplace{[ ]}{name\endcsname}}%
3132     \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
3133     \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3134     \ifin@
3135         \@nameuse{bbl@patch\bbl@tempa}%
3136         \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3137     \fi
3138     \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3139     \ifin@
3140         \toks@\expandafter{\bbl@toreplace}%
3141         \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}%
3142     \fi}
3143 \def\bbl@ini@captions@aux#1#2{%
3144     \bbl@trim@def\bbl@tempa{#1}%
3145     \bbl@xin@{.template}{\bbl@tempa}%
3146     \ifin@
3147         \bbl@ini@captions@template{#2}\language
3148     \else
3149         \bbl@ifblank{#2}%
3150             {\bbl@exp{%
3151                 \toks@{\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
3152                 {\bbl@trim\toks@{#2}}}%
3153             \bbl@exp{%
3154                 \bbl@add{\bbl@savestrings{%

```

```

3155      \\SetString\<\bbl@tempa name>{\the\toks@}}}%
3156 \toks@\expandafter{\bbl@captionslist}%
3157 \bbl@exp{\\in@{\<\bbl@tempa name>}{\the\toks@}}}%
3158 \ifin@else
3159   \bbl@exp{%
3160     \\bbl@add\<bbl@extracaps@\language name>{\<\bbl@tempa name>}%
3161     \\bbl@tglobal\<bbl@extracaps@\language name>}%
3162   \fi
3163 \fi}

```

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

```

3164 \def\bbl@list@the{%
3165   part,chapter,section,subsection,subsubsection,paragraph,%
3166   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3167   table,page,footnote,mpfootnote,mpfn}
3168 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3169   \bbl@ifunset{\bbl@map@#1@\language name}%
3170     {\@nameuse{#1}}}%
3171     {\@nameuse{\bbl@map@#1@\language name}}}%
3172 \def\bbl@inikv@labels#1#2{%
3173   \in@{.map}{#1}%
3174   \ifin@
3175     \ifx\bbl@KVP@labels\@nnil\else
3176       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3177       \ifin@
3178         \def\bbl@tempc{#1}%
3179         \bbl@replace\bbl@tempc{.map}{}%
3180         \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3181         \bbl@exp{%
3182           \gdef\<\bbl@map@\bbl@tempc @\language name>%
3183             {\ifin@<#2>\else\\localecounter{#2}\fi}}}%
3184         \bbl@foreach\bbl@list@the{%
3185           \bbl@ifunset{the##1}{}%
3186           {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3187             \bbl@exp{%
3188               \\bbl@sreplace\<the##1>%
3189               {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3190               \\bbl@sreplace\<the##1>%
3191               {\<\@empty @\bbl@tempc>\<c@##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3192             \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3193               \toks@\expandafter\expandafter\expandafter{%
3194                 \csname the##1\endcsname}%
3195               \expandafter\xdef\csname the##1\endcsname{\the\toks@}}}%
3196             \fi}}}%
3197   \fi
3198 \fi
3199 %
3200 \else
3201   %
3202   % The following code is still under study. You can test it and make
3203   % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3204   % language dependent.
3205   \in@{enumerate.}{#1}%
3206   \ifin@
3207     \def\bbl@tempa{#1}%
3208     \bbl@replace\bbl@tempa{enumerate.}{}%
3209     \def\bbl@toreplace{#2}%
3210     \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3211     \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3212     \bbl@replace\bbl@toreplace{[ ]}{\endcsname{}}}%
3213     \toks@\expandafter{\bbl@toreplace}%
3214     % TODO. Execute only once:
3215     \bbl@exp{%

```



```

3216      \\\bbl@add\<extras\language>{%
3217      \\\babel@save\<labelenum\romannumeral\bbl@tempa>%
3218      \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3219      \\\bbl@tglobal\<extras\language>}%
3220      \fi
3221      \fi}

```

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

```

3222 \def\bbl@chapttype{chapter}
3223 \ifx\@makechapterhead\@undefined
3224   \let\bbl@patchchapter\relax
3225 \else\ifx\thechapter\@undefined
3226   \let\bbl@patchchapter\relax
3227 \else\ifx\ps@headings\@undefined
3228   \let\bbl@patchchapter\relax
3229 \else
3230   \def\bbl@patchchapter{%
3231     \global\let\bbl@patchchapter\relax
3232     \gdef\bbl@chfmt{%
3233       \bbl@ifunset{\bbl@bbl@chapttype fmt@\language}%
3234       {\@chapapp\space\thechapter}
3235       {\@nameuse{\bbl@bbl@chapttype fmt@\language}}}%
3236   \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3237   \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3238   \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3239   \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3240   \bbl@tglobal\appendix
3241   \bbl@tglobal\ps@headings
3242   \bbl@tglobal\chaptermark
3243   \bbl@tglobal\@makechapterhead}
3244   \let\bbl@patchappendix\bbl@patchchapter
3245 \fi\fi\fi
3246 \ifx\@part\@undefined
3247   \let\bbl@patchpart\relax
3248 \else
3249   \def\bbl@patchpart{%
3250     \global\let\bbl@patchpart\relax
3251     \gdef\bbl@partformat{%
3252       \bbl@ifunset{\bbl@partfmt@\language}%
3253       {\partname\nobreakspace\thepart}
3254       {\@nameuse{\bbl@partfmt@\language}}}%
3255   \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3256   \bbl@tglobal\@part}
3257 \fi

```

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

```

3258 \let\bbl@calendar\@empty
3259 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3260 \def\bbl@localedate#1#2#3#4{%
3261   \begingroup
3262     \edef\bbl@they{#2}%
3263     \edef\bbl@them{#3}%
3264     \edef\bbl@thed{#4}%
3265     \edef\bbl@tempe{%
3266       \bbl@ifunset{\bbl@calpr@\language}}{\bbl@cl{calpr}},%
3267       #1}%
3268   \bbl@replace\bbl@tempe{ }{}%
3269   \bbl@replace\bbl@tempe{CONVERT}{convert=}% Hackish
3270   \bbl@replace\bbl@tempe{convert}{convert=}%
3271   \let\bbl@ld@calendar\@empty

```

```

3272 \let\bbl@ld@variant\@empty
3273 \let\bbl@ld@convert\relax
3274 \def\bbl@tempb##1=##2@@{\@namedef\bbl@ld@##1}{##2}}%
3275 \bbl@foreach\bbl@tempe{\bbl@tempb##1\@}%
3276 \bbl@replace\bbl@ld@calendar{gregorian}{}%
3277 \ifx\bbl@ld@calendar\@empty\else
3278   \ifx\bbl@ld@convert\relax\else
3279     \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3280     {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3281   \fi
3282 \fi
3283 \@nameuse\bbl@precalendar}% Remove, eg, +, -civil (-ca-islamic)
3284 \edef\bbl@calendar{% Used in \month..., too
3285   \bbl@ld@calendar
3286   \ifx\bbl@ld@variant\@empty\else
3287     .\bbl@ld@variant
3288   \fi}%
3289 \bbl@cased
3290   {\@nameuse\bbl@date@\language@name @\bbl@calendar}%
3291   \bbl@they\bbl@them\bbl@thed}%
3292 \endgroup}
3293 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3294 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3295   \bbl@trim@def\bbl@tempa{#1.#2}%
3296   \bbl@ifsamestring{\bbl@tempa}{months.wide}%      to savedate
3297   {\bbl@trim@def\bbl@tempa{#3}%
3298     \bbl@trim\toks@{#5}%
3299     \@temptokena\expandafter{\bbl@savedate}%
3300     \bbl@exp{% Reverse order - in ini last wins
3301       \def\\bbl@savedate{%
3302         \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3303         \the\@temptokena}}}%
3304     {\bbl@ifsamestring{\bbl@tempa}{date.long}%      defined now
3305       {\lowercase{\def\bbl@tempb{#6}}}%
3306       \bbl@trim@def\bbl@toreplace{#5}%
3307       \bbl@TG@@date
3308       \global\bbl@csarg\let{date@\language@name @\bbl@tempb}\bbl@toreplace
3309       \ifx\bbl@savetoday\@empty
3310         \bbl@exp{% TODO. Move to a better place.
3311           \\AfterBabelCommands{%
3312             \def\<\language@name date>{\\protect\<\language@name date >}%
3313             \\newcommand\<\language@name date >[4][]{%
3314               \\bbl@usedategroupttrue
3315               \<bbl@ensure@\language@name>{%
3316                 \\localedate[####1]{####2}{####3}{####4}}}%
3317             \def\\bbl@savetoday{%
3318               \\SetString\\today{%
3319                 \<\language@name date>[convert]%
3320                 {\the\year}{\the\month}{\the\day}}}%
3321             \fi}%
3322             {}}}}

```

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

```

3323 \let\bbl@calendar\@empty
3324 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3325   \@nameuse\bbl@ca@#2}#1\@}%
3326 \newcommand\babelDateSpace{\nobreakspace}
3327 \newcommand\babelDateDot{.\@} % TODO. \let instead of repeating
3328 \newcommand\babelDated[1]{\number#1}%

```

```

3329 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3330 \newcommand\BabelDateM[1]{\number#1}
3331 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3332 \newcommand\BabelDateMMM[1]{%
3333   \csname month\romannumeral#1\bbbl@calendar name\endcsname}%
3334 \newcommand\BabelDatey[1]{\number#1}%
3335 \newcommand\BabelDateyy[1]{%
3336   \ifnum#1<10 0\number#1 %
3337   \else\ifnum#1<100 \number#1 %
3338   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3339   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3340   \else
3341     \bbbl@error
3342     {Currently two-digit years are restricted to the\
3343       range 0-9999.}%
3344     {There is little you can do. Sorry.}%
3345   \fi\fi\fi\fi}
3346 \newcommand\BabelDateyyyy[1]{\number#1} % TODO - add leading 0
3347 \def\bbbl@replace@finish@iii#1{%
3348   \bbbl@exp{\def\#1###1###2###3{\the\toks@}}
3349 \def\bbbl@TG@date{%
3350   \bbbl@replace\bbbl@toreplace{[ ]}{\BabelDateSpace{}}%
3351   \bbbl@replace\bbbl@toreplace{[.]}{\BabelDateDot{}}%
3352   \bbbl@replace\bbbl@toreplace{[d]}{\BabelDated{###3}}%
3353   \bbbl@replace\bbbl@toreplace{[dd]}{\BabelDatedd{###3}}%
3354   \bbbl@replace\bbbl@toreplace{[M]}{\BabelDateM{###2}}%
3355   \bbbl@replace\bbbl@toreplace{[MM]}{\BabelDateMM{###2}}%
3356   \bbbl@replace\bbbl@toreplace{[MMM]}{\BabelDateMMM{###2}}%
3357   \bbbl@replace\bbbl@toreplace{[y]}{\BabelDatey{###1}}%
3358   \bbbl@replace\bbbl@toreplace{[yy]}{\BabelDateyy{###1}}%
3359   \bbbl@replace\bbbl@toreplace{[yyy]}{\BabelDateyyy{###1}}%
3360   \bbbl@replace\bbbl@toreplace{[y|]}{\bbbl@datecctr[###1|]}%
3361   \bbbl@replace\bbbl@toreplace{[m|]}{\bbbl@datecctr[###2|]}%
3362   \bbbl@replace\bbbl@toreplace{[d|]}{\bbbl@datecctr[###3|]}%
3363   \bbbl@replace@finish@iii\bbbl@toreplace}
3364 \def\bbbl@datecctr{\expandafter\bbbl@xdatecctr\expandafter}
3365 \def\bbbl@xdatecctr[#1|#2]{\localenumeral{#2}{#1}}

```

### Transforms.

```

3366 \let\bbbl@release@transforms\empty
3367 \bbbl@csarg\let{inikv@transforms.prehyphenation}\bbbl@inikv
3368 \bbbl@csarg\let{bbbl@inikv@transforms.posthyphenation}\bbbl@inikv
3369 \def\bbbl@transforms@aux#1#2#3#4,#5\relax{%
3370   #1[#2]{#3}{#4}{#5}}
3371 \begin{group} % A hack. TODO. Don't require an specific order
3372   \catcode`\=12
3373   \catcode`\&=14
3374   \def\bbbl@transforms#1#2#3{%&
3375     \directlua{
3376       local str = [=[#2]=]
3377       str = str:gsub('%.%d+%.%d+$', '')
3378       tex.print([[def\string\babeltempa{]] .. str .. [[]])
3379     }&
3380     \bbbl@xin@{,\babeltempa,}{,\bbbl@KVP@transforms,}&
3381     \ifin@
3382       \in@{.0$}{#2$}&
3383     \ifin@
3384       \directlua{& (\attribute) syntax
3385         local str = string.match([[ \bbbl@KVP@transforms]],
3386           '%([^(%-)%)^[^)]-\babeltempa')
3387         if str == nil then
3388           tex.print([[ \def\string\babeltempb{]])
3389         else

```

```

3390         tex.print([[\\def\\string\\babeltempb{,attribute=} .. str .. [{}]])
3391     end
3392 }
3393 \\toks@{#3}&%
3394 \\bbl@exp{&%
3395     \\g@addto@macro\\bbl@release@transforms{&%
3396         \\relax &% Closes previous \\bbl@transforms@aux
3397         \\bbl@transforms@aux
3398         \\#1{label=\\babeltempa\\babeltempb}{\\language\\name}{\\the\\toks@}}&%
3399     \\else
3400         \\g@addto@macro\\bbl@release@transforms{, {#3}}&%
3401     \\fi
3402 \\fi}
3403 \\endgroup

```

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

```

3404 \\def\\bbl@provide@lsys#1{%
3405     \\bbl@ifunset{bbl@lname@#1}%
3406     {\\bbl@load@info{#1}}%
3407     {}%
3408     \\bbl@csarg\\let{lsys@#1}\\@empty
3409     \\bbl@ifunset{bbl@sname@#1}{\\bbl@csarg\\gdef{sname@#1}{Default}}{}%
3410     \\bbl@ifunset{bbl@sotf@#1}{\\bbl@csarg\\gdef{sotf@#1}{DFLT}}{}%
3411     \\bbl@csarg\\bbl@add@list{lsys@#1}{Script=\\bbl@cs{sname@#1}}%
3412     \\bbl@ifunset{bbl@lname@#1}{}%
3413     {\\bbl@csarg\\bbl@add@list{lsys@#1}{Language=\\bbl@cs{lname@#1}}}%
3414     \\ifcase\\bbl@engine\\or\\or
3415         \\bbl@ifunset{bbl@prehc@#1}{}%
3416         {\\bbl@exp{\\bbl@ifblank{\\bbl@cs{prehc@#1}}}%
3417             {}%
3418             {\\ifx\\bbl@xenohyph\\@undefined
3419                 \\global\\let\\bbl@xenohyph\\bbl@xenohyph@d
3420                 \\ifx\\AtBeginDocument\\@notprerr
3421                     \\expandafter\\@secondoftwo % to execute right now
3422                     \\fi
3423                     \\AtBeginDocument{%
3424                         \\bbl@patchfont{\\bbl@xenohyph}%
3425                         \\expandafter\\selectlanguage\\expandafter{\\language\\name}}%
3426                     \\fi}%
3427             \\fi
3428             \\bbl@csarg\\bbl@toglobal{lsys@#1}}
3429 \\def\\bbl@xenohyph@d{%
3430     \\bbl@ifset{bbl@prehc@\\language\\name}%
3431     {\\ifnum\\hyphenchar\\font=\\defaultthyphenchar
3432         \\iffontchar\\font\\bbl@cl{prehc}\\relax
3433         \\hyphenchar\\font\\bbl@cl{prehc}\\relax
3434     \\else\\iffontchar\\font"200B
3435         \\hyphenchar\\font"200B
3436     \\else
3437         \\bbl@warning
3438         {Neither 0 nor ZERO WIDTH SPACE are available\\%
3439             in the current font, and therefore the hyphen\\%
3440             will be printed. Try changing the fontspec's\\%
3441             'HyphenChar' to another value, but be aware\\%
3442             this setting is not safe (see the manual).\\%
3443             Reported}%
3444         \\hyphenchar\\font\\defaultthyphenchar
3445         \\fi\\fi
3446     \\fi}%
3447     {\\hyphenchar\\font\\defaultthyphenchar}}
3448     % \\fi}

```

The following ini reader ignores everything but the identification section. It is called when a font

is defined (ie, when the language is first selected) to know which script/language must be enabled. This means we must make sure a few characters are not active. The ini is not read directly, but with a proxy tex file named as the language (which means any code in it must be skipped, too).

```

3449 \def\bbl@load@info#1{%
3450   \def\BabelBeforeIni##1##2{%
3451     \begingroup
3452       \bbl@read@ini{##1}0%
3453       \endinput           % babel- .tex may contain onlypreamble's
3454       \endgroup}%         boxed, to avoid extra spaces:
3455   {\bbl@input@texini{#1}}}
```

A tool to define the macros for native digits from the list provided in the ini file. Somewhat convoluted because there are 10 digits, but only 9 arguments in T<sub>E</sub>X. Non-digits characters are kept. The first macro is the generic “localized” command.

```

3456 \def\bbl@setdigits#1#2#3#4#5{%
3457   \bbl@exp{%
3458     \def<\language name digits>####1{%          ie, \langdigits
3459       \<bbl@digits@\language name>####1\\\nil}%
3460       \let<\bbl@cntr@digits@\language name>\<\language name digits>%
3461       \def<\language name counter>####1{%        ie, \langcounter
3462         \expandafter\<bbl@counter@\language name>%
3463         \\\csname c#####1\endcsname}%
3464         \def<\bbl@counter@\language name>####1{% ie, \bbl@counter@lang
3465           \expandafter\<bbl@digits@\language name>%
3466           \\\number####1\\\nil}}}%
3467   \def\bbl@tempa####1##2##3##4##5{%
3468     \bbl@exp{% Wow, quite a lot of hashes! :- (
3469       \def<\bbl@digits@\language name>#####1{%
3470         \\\ifx#####1\\\nil           % ie, \bbl@digits@lang
3471         \\\else
3472           \\\ifx0#####1#1%
3473           \\\else\\\ifx1#####1#2%
3474           \\\else\\\ifx2#####1#3%
3475           \\\else\\\ifx3#####1#4%
3476           \\\else\\\ifx4#####1#5%
3477           \\\else\\\ifx5#####1##1%
3478           \\\else\\\ifx6#####1##2%
3479           \\\else\\\ifx7#####1##3%
3480           \\\else\\\ifx8#####1##4%
3481           \\\else\\\ifx9#####1##5%
3482           \\\else#####1%
3483           \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
3484           \expandafter\<bbl@digits@\language name>%
3485           \\\fi}}}%
3486   \bbl@tempa}
```

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

```

3487 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={ }
3488   \ifx\\\#1%           % \ before, in case #1 is multiletter
3489     \bbl@exp{%
3490       \def\\\bbl@tempa####1{%
3491         \ifcase>####1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
3492       \else
3493         \toks@\expandafter{\the\toks@\or #1}%
3494         \expandafter\bbl@buildifcase
3495       \fi}
```

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

```

3496 \newcommand\localnumeral[2]{\bbl@cs{cntr@#1@\language name}{#2}}
```

```

3497 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3498 \newcommand\localecounter[2]{%
3499   \expandafter\bbl@localecntr
3500   \expandafter{\number\csname c@#2\endcsname}{#1}}
3501 \def\bbl@alphnumeral#1#2{%
3502   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3503 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3504   \ifcase\@car#8\@nil\or    % Currenty <10000, but prepared for bigger
3505     \bbl@alphnumeral@ii{#9}000000#1\or
3506     \bbl@alphnumeral@ii{#9}00000#1#2\or
3507     \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3508     \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3509     \bbl@alphnum@invalid{>9999}%
3510   \fi}
3511 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3512   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\language}%
3513     {\bbl@cs{cntr@#1.4@\language}#5%
3514     \bbl@cs{cntr@#1.3@\language}#6%
3515     \bbl@cs{cntr@#1.2@\language}#7%
3516     \bbl@cs{cntr@#1.1@\language}#8%
3517     \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3518       \bbl@ifunset{bbl@cntr@#1.S.321@\language}{}%
3519       {\bbl@cs{cntr@#1.S.321@\language}}%
3520     \fi}%
3521   {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}}}
3522 \def\bbl@alphnum@invalid#1{%
3523   \bbl@error{Alphabetic numeral too large (#1)}%
3524   {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.

```

3525 \def\bbl@localeinfo#1#2{%
3526   \bbl@ifunset{bbl@info@#2}{#1}%
3527     {\bbl@ifunset{bbl@\csname bbl@info@#2\endcsname @\language}{#1}%
3528     {\bbl@cs{\csname bbl@info@#2\endcsname @\language}}}%
3529 \newcommand\localeinfo[1]{%
3530   \ifx*#1@empty    % TODO. A bit hackish to make it expandable.
3531     \bbl@afterelse\bbl@localeinfo{}%
3532   \else
3533     \bbl@localeinfo
3534     {\bbl@error{I've found no info for the current locale.\%
3535       The corresponding ini file has not been loaded\\%
3536       Perhaps it doesn't exist}%
3537     {See the manual for details.}}%
3538   {#1}%
3539   \fi}
3540 % \@namedef{bbl@info@name.locale}{lname}
3541 \@namedef{bbl@info@tag.ini}{lini}
3542 \@namedef{bbl@info@name.english}{elname}
3543 \@namedef{bbl@info@name.opentype}{lname}
3544 \@namedef{bbl@info@tag.bcp47}{tbc}
3545 \@namedef{bbl@info@language.tag.bcp47}{lbcp}
3546 \@namedef{bbl@info@tag.opentype}{lotf}
3547 \@namedef{bbl@info@script.name}{esname}
3548 \@namedef{bbl@info@script.name.opentype}{sname}
3549 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3550 \@namedef{bbl@info@script.tag.opentype}{sotf}
3551 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3552 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3553 % Extensions are dealt with in a special way
3554 % Now, an internal \LaTeX{} macro:
3555 \providecommand\BCPdata[1]{\localeinfo*{#1.tag.bcp47}}

```

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

```

3556 <<{*More package options}> ≡
3557 \DeclareOption{ensureinfo=off}{}
3558 <</More package options>>
3559 %
3560 \let\bbl@ensureinfo\@gobble
3561 \newcommand\BabelEnsureInfo{%
3562   \ifx\InputIfFileExists\undefined\else
3563     \def\bbl@ensureinfo##1{%
3564       \bbl@ifunset{\bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3565   \fi
3566   \bbl@foreach\bbl@loaded{%
3567     \def\language{##1}%
3568     \bbl@ensureinfo{##1}}}%
3569 \ifpackagewith{babel}{ensureinfo=off}{}%
3570 {\AtEndOfPackage{% Test for plain.
3571   \ifx\undefined\bbl@loaded\else\BabelEnsureInfo\fi}}

```

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

```

3572 \newcommand\getlocaleproperty{%
3573   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3574 \def\bbl@getproperty@s#1#2#3{%
3575   \let#1\relax
3576   \def\bbl@elt##1##2##3{%
3577     \bbl@ifsamestring{##1/##2}{##3}%
3578     {\providecommand#1{##3}%
3579     \def\bbl@elt####1####2####3{}}}%
3580   }%
3581   \bbl@cs{inidata@#2}}%
3582 \def\bbl@getproperty@x#1#2#3{%
3583   \bbl@getproperty@s{#1}{#2}{#3}%
3584   \ifx#1\relax
3585     \bbl@error
3586       {Unknown key for locale '#2':\%
3587       #3\}%
3588     \string#1 will be set to \relax}%
3589   {Perhaps you misspelled it.}%
3590   \fi}
3591 \let\bbl@ini@loaded\@empty
3592 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

## 8 Adjusting the Babel bahavior

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

```

3593 \newcommand\babeladjust[1]{% TODO. Error handling.
3594   \bbl@forkv{#1}{%
3595     \bbl@ifunset{\bbl@ADJ@##1@##2}%
3596     {\bbl@cs{ADJ@##1}{##2}}%
3597     {\bbl@cs{ADJ@##1@##2}}}
3598 %
3599 \def\bbl@adjust@lua#1#2{%
3600   \ifvmode
3601     \ifnum\currentgrouplevel=\z@
3602       \directlua{ Babel.#2 }%
3603       \expandafter\expandafter\expandafter\@gobble
3604     \fi
3605   \fi
3606   {\bbl@error % The error is gobbled if everything went ok.
3607     {Currently, #1 related features can be adjusted only\%
3608     in the main vertical list.}%
3609     {Maybe things change in the future, but this is what it is.}}}

```

```

3610 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3611   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3612 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3613   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3614 \@namedef{bbl@ADJ@bidi.text@on}{%
3615   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3616 \@namedef{bbl@ADJ@bidi.text@off}{%
3617   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3618 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3619   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3620 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3621   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3622 %
3623 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3624   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3625 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3626   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3627 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3628   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3629 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3630   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3631 \@namedef{bbl@ADJ@justify.arabic@on}{%
3632   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3633 \@namedef{bbl@ADJ@justify.arabic@off}{%
3634   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3635 %
3636 \def\bbl@adjust@layout#1{%
3637   \ifvmode
3638     #1%
3639     \expandafter\@gobble
3640   \fi
3641   {\bbl@error % The error is gobbled if everything went ok.
3642     {Currently, layout related features can be adjusted only\\%
3643       in vertical mode.}%
3644     {Maybe things change in the future, but this is what it is.}}}
3645 \@namedef{bbl@ADJ@layout.tabular@on}{%
3646   \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}}
3647 \@namedef{bbl@ADJ@layout.tabular@off}{%
3648   \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}}
3649 \@namedef{bbl@ADJ@layout.lists@on}{%
3650   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3651 \@namedef{bbl@ADJ@layout.lists@off}{%
3652   \bbl@adjust@layout{\let\list\bbl@OL@list}}
3653 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
3654   \bbl@activateposthyphen}
3655 %
3656 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3657   \bbl@bcpallowedtrue}
3658 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3659   \bbl@bcpallowedfalse}
3660 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3661   \def\bbl@bcp@prefix{#1}}
3662 \def\bbl@bcp@prefix{bcp47-}
3663 \@namedef{bbl@ADJ@autoload.options}#1{%
3664   \def\bbl@autoload@options{#1}}
3665 \let\bbl@autoload@bcptoptions\@empty
3666 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3667   \def\bbl@autoload@bcptoptions{#1}}
3668 \newif\ifbbl@bcptoname
3669 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3670   \bbl@bcptonametrue}
3671 \BabelEnsureInfo}
3672 \@namedef{bbl@ADJ@bcp47.toname@off}{%

```



```

3673 \bbl@bcptonamefalse}
3674 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3675 \directlua{ Babel.ignore_pre_char = function(node)
3676     return (node.lang == \the\csname l@nohyphenation\endcsname)
3677 end }}
3678 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3679 \directlua{ Babel.ignore_pre_char = function(node)
3680     return false
3681 end }}
3682 \@namedef{bbl@ADJ@select.write@shift}{%
3683 \let\bbl@restorelastskip\relax
3684 \def\bbl@savelastskip{%
3685 \let\bbl@restorelastskip\relax
3686 \ifvmode
3687 \ifdim\lastskip=\z@
3688 \let\bbl@restorelastskip\nobreak
3689 \else
3690 \bbl@exp{%
3691 \def\\bbl@restorelastskip{%
3692 \skip@=\the\lastskip
3693 \\nobreak \vskip-\skip@ \vskip\skip@}}%
3694 \fi
3695 \fi}}
3696 \@namedef{bbl@ADJ@select.write@keep}{%
3697 \let\bbl@restorelastskip\relax
3698 \let\bbl@savelastskip\relax}
3699 \@namedef{bbl@ADJ@select.write@omit}{%
3700 \let\bbl@restorelastskip\relax
3701 \def\bbl@savelastskip##1\bbl@restorelastskip{}}

```

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

```

3702 \ifx\directlua\@undefined\else
3703 \ifx\bbl@luapatterns\@undefined
3704 \input luababel.def
3705 \fi
3706 \fi

```

Continue with  $\LaTeX$ .

```

3707 </package | core>
3708 <*package>

```

## 8.1 Cross referencing macros

The  $\LaTeX$  book states:

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

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

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

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

```

3709 <(*More package options)> ≡
3710 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3711 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3712 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3713 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3714 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3715 <(/More package options)>

```

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

```

3716 \bbl@trace{Cross referencing macros}
3717 \ifx\bbl@opt@safe\empty\else % ie, if 'ref' and/or 'bib'
3718   \def\@newl@bel#1#2#3{%
3719     {\@safe@activestrue
3720       \bbl@ifunset{#1@#2}%
3721       \relax
3722       {\gdef\@multiplelabels{%
3723         \@latex@warning@no@line{There were multiply-defined labels}}}%
3724         \@latex@warning@no@line{Label `#2' multiply defined}}}%
3725   \global\@namedef{#1@#2}{#3}}

```

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

```

3726   \CheckCommand*\@testdef[3]{%
3727     \def\reserved@a{#3}%
3728     \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3729     \else
3730       \@tempswatrue
3731     \fi}

```

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

```

3732   \def\@testdef#1#2#3{% TODO. With @samestring?
3733     \@safe@activestrue
3734     \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3735     \def\bbl@tempb{#3}%
3736     \@safe@activesfalse
3737     \ifx\bbl@tempa\relax
3738     \else
3739       \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3740     \fi
3741     \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3742     \ifx\bbl@tempa\bbl@tempb
3743     \else
3744       \@tempswatrue
3745     \fi}
3746 \fi

```

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

```

3747 \bbl@xin@{R}\bbl@opt@safe
3748 \ifin@
3749   \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3750   \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3751   {\expandafter\strip@prefix\meaning\ref}%
3752   \ifin@
3753     \bbl@redefine\@kernel@ref#1{%
3754       \@safe@activestrue\org@@kernel@ref{#1}\@safe@activesfalse}
3755     \bbl@redefine\@kernel@pageref#1{%
3756       \@safe@activestrue\org@@kernel@pageref{#1}\@safe@activesfalse}
3757     \bbl@redefine\@kernel@sref#1{%
3758       \@safe@activestrue\org@@kernel@sref{#1}\@safe@activesfalse}
3759     \bbl@redefine\@kernel@spageref#1{%
3760       \@safe@activestrue\org@@kernel@spageref{#1}\@safe@activesfalse}
3761   \else
3762     \bbl@redefineroobust\ref#1{%
3763       \@safe@activestrue\org@ref{#1}\@safe@activesfalse}
3764     \bbl@redefineroobust\pageref#1{%
3765       \@safe@activestrue\org@pageref{#1}\@safe@activesfalse}

```

```

3766 \fi
3767 \else
3768 \let\org@ref\ref
3769 \let\org@pageref\pageref
3770 \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.

```

3771 \bbl@xin@{B}\bbl@opt@safe
3772 \ifin@
3773 \bbl@redefine\@citex[#1]#2{%
3774 \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
3775 \org@@citex[#1]{\@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.

```

3776 \AtBeginDocument{%
3777 \ifpackageloaded{natbib}{%

```

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

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

```

3778 \def\@citex[#1][#2]#3{%
3779 \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
3780 \org@@citex[#1][#2]{\@tempa}}%
3781 }{}

```

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

```

3782 \AtBeginDocument{%
3783 \ifpackageloaded{cite}{%
3784 \def\@citex[#1]#2{%
3785 \@safe@activetrue\org@@citex[#1][#2]\@safe@activesfalse}%
3786 }{}

```

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

```

3787 \bbl@redefine\nocite#1{%
3788 \@safe@activetrue\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.

```

3789 \bbl@redefine\bibcite{%
3790 \bbl@cite@choice
3791 \bibcite}

```

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

```

3792 \def\bbl@bibcite#1#2{%
3793 \org@bibcite{#1}{\@safe@activesfalse#2}}

```

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

```

3794 \def\bbl@cite@choice{%
3795   \global\let\bibcite\bbl@bibcite
3796   \ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
3797   \ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
3798   \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.

```

3799 \AtBeginDocument{\bbl@cite@choice}

```

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

```

3800 \bbl@redefine\@bibitem#1{%
3801   \@safe@activetrue\org@@bibitem{#1}\@safe@activesfalse}
3802 \else
3803   \let\org@nocite\nocite
3804   \let\org@@citex\@citex
3805   \let\org@bibcite\bibcite
3806   \let\org@@bibitem\@bibitem
3807 \fi

```

## 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 bidi documents.

```

3808 \bbl@trace{Marks}
3809 \IfBabelLayout{sectioning}
3810   {\ifx\bbl@opt@headfoot\@nnil
3811     \g@addto@macro\resetactivechars{%
3812       \set@typeset@protect
3813       \expandafter\select@language@x\expandafter{\bbl@main@language}%
3814       \let\protect\noexpand
3815       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3816         \edef\thepage{%
3817           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3818       \fi}%
3819   \fi}
3820 {\ifbbl@single\else
3821   \bbl@ifunset{markright }{\bbl@redefine\bbl@redefineroobust
3822     \markright#1{%
3823       \bbl@ifblank{#1}%
3824       {\org@markright{}}}%
3825       {\toks@{#1}%
3826        \bbl@exp{%
3827          \org@markright{\protect\foreignlanguage{\language}\thepage}%
3828          {\protect\bbl@restore@actives\the\toks@}}}%

```

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

```

3829   \ifx\@mkboth\markboth
3830     \def\bbl@tempc{\let\@mkboth\markboth}
3831   \else
3832     \def\bbl@tempc{}
3833   \fi
3834   \bbl@ifunset{markboth }{\bbl@redefine\bbl@redefineroobust

```

```

3835 \markboth#1#2{%
3836 \protected@edef\bbl@tempb##1{%
3837 \protect\foreignlanguage
3838 {\language\name}\protect\bbl@restore@actives##1}}%
3839 \bbl@ifblank{#1}%
3840 {\toks@{}}%
3841 {\toks@\expandafter{\bbl@tempb{#1}}}%
3842 \bbl@ifblank{#2}%
3843 {\@temptokena{}}%
3844 {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3845 \bbl@exp{\org@markboth{\the\toks@}{\the\@temptokena}}
3846 \bbl@tempc
3847 \fi} % end ifbbl@single, end \IfBabelLayout

```

## 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.

```

3848 \bbl@trace{Preventing clashes with other packages}
3849 \ifx\org@ref\undefined\else
3850 \bbl@xin@{R}\bbl@opt@safe
3851 \ifin@
3852 \AtBeginDocument{%
3853 \ifpackageloaded{ifthen}{%
3854 \bbl@redefine@long\ifthenelse#1#2#3{%
3855 \let\bbl@temp@pref\pageref
3856 \let\pageref\org@pageref
3857 \let\bbl@temp@ref\ref
3858 \let\ref\org@ref
3859 \@safe@activestrue
3860 \org@ifthenelse{#1}%
3861 {\let\pageref\bbl@temp@pref
3862 \let\ref\bbl@temp@ref
3863 \@safe@activesfalse
3864 #2}%
3865 {\let\pageref\bbl@temp@pref
3866 \let\ref\bbl@temp@ref
3867 \@safe@activesfalse
3868 #3}%
3869 }%
3870 }}%
3871 }
3872 \fi

```

### 8.3.2 varioref

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

happen for \vrefpagenum.

```

3873 \AtBeginDocument{%
3874   \@ifpackageloaded{varioref}{%
3875     \bbl@redefine\@vpageref#1[#2]#3{%
3876       \@safe@activetrue
3877       \org@@vpageref{#1}[#2]#3}%
3878     \@safe@activesfalse}%
3879   \bbl@redefine\vrefpagenum#1#2{%
3880     \@safe@activetrue
3881     \org\vrefpagenum{#1}#2}%
3882   \@safe@activesfalse}%

```

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

```

3883   \expandafter\def\csname Ref \endcsname#1{%
3884     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3885   }{}%
3886 }
3887 \fi

```

### 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.

```

3888 \AtEndOfPackage{%
3889   \AtBeginDocument{%
3890     \@ifpackageloaded{hhline}%
3891     {\expandafter\ifx\csname normal@char\string\endcsname\relax
3892       \else
3893         \makeatletter
3894         \def\@currname{hhline}\input{hhline.sty}\makeatother
3895       \fi}%
3896     {}}

```

\substitutefontfamily Deprecated. Use the tools provides by L<sup>A</sup>T<sub>E</sub>X. 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.

```

3897 \def\substitutefontfamily#1#2#3{%
3898   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3899   \immediate\write15{%
3900     \string\ProvidesFile{#1#2.fd}%
3901     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3902     \space generated font description file]^J
3903     \string\DeclareFontFamily{#1}{#2}{}^^J
3904     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
3905     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
3906     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
3907     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
3908     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
3909     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
3910     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
3911     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
3912   }%
3913   \closeout15
3914 }
3915 \@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`

```
3916 \bbl@trace{Encoding and fonts}
3917 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3918 \newcommand\BabelNonText{TS1,T3,TS3}
3919 \let\org@TeX\TeX
3920 \let\org@LaTeX\LaTeX
3921 \let\ensureascii\@firstofone
3922 \AtBeginDocument{%
3923   \def\@elt#1{,#1,}%
3924   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3925   \let\@elt\relax
3926   \let\bbl@tempb\@empty
3927   \def\bbl@tempc{OT1}%
3928   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3929     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3930   \bbl@foreach\bbl@tempa{%
3931     \bbl@xin@{#1}{\BabelNonASCII}%
3932     \ifin@
3933       \def\bbl@tempb{#1}% Store last non-ascii
3934     \else\bbl@xin@{#1}{\BabelNonText}% Pass
3935       \ifin@
3936       \def\bbl@tempc{#1}% Store last ascii
3937     \fi
3938   \fi}%
3939   \ifx\bbl@tempb\@empty\else
3940     \bbl@xin@{\cf@encoding,}{\BabelNonASCII,\BabelNonText,}%
3941     \ifin@
3942     \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3943     \fi
3944     \edef\ensureascii#1{%
3945       {\noexpand\fontencoding{\bbl@tempc}\noexpand\selectfont#1}}%
3946     \DeclareTextCommandDefault{\TeX}{\ensureascii\org@TeX}%
3947     \DeclareTextCommandDefault{\LaTeX}{\ensureascii\org@LaTeX}%
3948   \fi}
```

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

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

```
3949 \AtEndOfPackage{\edef\latinencoding{\cf@encoding}}
```

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

```
3950 \AtBeginDocument{%
3951   \ifpackageloaded{fontspec}%
3952     {\xdef\latinencoding{%
3953       \ifx\UTFencname\@undefined
3954         EU\ifcase\bbl@engine\or2\or1\fi
3955       \else
3956         \UTFencname
3957       \fi}}%
3958   {\gdef\latinencoding{OT1}%
```

```

3959 \ifx\cf@encoding\bbl@t@one
3960 \xdef\latinencoding{\bbl@t@one}%
3961 \else
3962 \def\elt#1{, #1,}%
3963 \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3964 \let\elt\relax
3965 \bbl@xin@{, T1, }\bbl@tempa
3966 \ifin@
3967 \xdef\latinencoding{\bbl@t@one}%
3968 \fi
3969 \fi}}

```

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

```

3970 \DeclareRobustCommand{\latintext}{%
3971 \fontencoding{\latinencoding}\selectfont
3972 \def\encodingdefault{\latinencoding}}

```

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

```

3973 \ifx\@undefined\DeclareTextFontCommand
3974 \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3975 \else
3976 \DeclareTextFontCommand{\textlatin}{\latintext}
3977 \fi

```

For several functions, we need to execute some code with `\selectfont`. With  $\text{\LaTeX}$  2021-06-01, there is a hook for this purpose, but in older versions the  $\text{\LaTeX}$  command is patched (the latter solution will be eventually removed).

```

3978 \def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}

```

## 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  $\text{\TeX}$  grouping.
- `luatex` can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but bidi text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As `Lua $\text{\TeX}$ -ja` shows, vertical typesetting is possible, too.

```

3979 \bbl@trace{Loading basic (internal) bidi support}
3980 \ifodd\bbl@engine
3981 \else % TODO. Move to txtbabel
3982 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3983 \bbl@error
3984 {The bidi method 'basic' is available only in\%
3985 luatex. I'll continue with 'bidi=default', so\%
3986 expect wrong results}%
3987 {See the manual for further details.}%

```



```

3988 \let\bbl@beforeforeign\leavevmode
3989 \AtEndOfPackage{%
3990 \EnableBabelHook{babel-bidi}%
3991 \bbl@xebidipar}
3992 \fi\fi
3993 \def\bbl@loadxebidi#1{%
3994 \ifx\RTLfootnotetext\@undefined
3995 \AtEndOfPackage{%
3996 \EnableBabelHook{babel-bidi}%
3997 \bbl@loadfontspec % bidi needs fontspec
3998 \usepackage#1{bidi}}%
3999 \fi}
4000 \ifnum\bbl@bidimode>200
4001 \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
4002 \bbl@tentative{bidi=bidi}
4003 \bbl@loadxebidi{}
4004 \or
4005 \bbl@loadxebidi{[rldocument]}
4006 \or
4007 \bbl@loadxebidi{}
4008 \fi
4009 \fi
4010 \fi
4011 % TODO? Separate:
4012 \ifnum\bbl@bidimode=\@ne
4013 \let\bbl@beforeforeign\leavevmode
4014 \ifodd\bbl@engine
4015 \newattribute\bbl@attr@dir
4016 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
4017 \bbl@exp{\output{\bodydir\pagedir\the\output}}
4018 \fi
4019 \AtEndOfPackage{%
4020 \EnableBabelHook{babel-bidi}%
4021 \ifodd\bbl@engine\else
4022 \bbl@xebidipar
4023 \fi}
4024 \fi

```

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

```

4025 \bbl@trace{Macros to switch the text direction}
4026 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
4027 \def\bbl@rscripts{% TODO. Base on codes ??
4028 ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
4029 Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
4030 Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
4031 Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
4032 Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
4033 Old South Arabian,}%
4034 \def\bbl@provide@dirs#1{%
4035 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
4036 \ifin@
4037 \global\bbl@csarg\chardef{wdir@#1}\@ne
4038 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4039 \ifin@
4040 \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
4041 \fi
4042 \else
4043 \global\bbl@csarg\chardef{wdir@#1}\z@
4044 \fi
4045 \ifodd\bbl@engine
4046 \bbl@csarg\ifcase{wdir@#1}%
4047 \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%

```

```

4048 \or
4049 \directlua{ Babel.locale_props[\the\localeid].texdir = 'r' }%
4050 \or
4051 \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
4052 \fi
4053 \fi}
4054 \def\bbl@switchdir{%
4055 \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4056 \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
4057 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}%
4058 \def\bbl@setdirs#1{% TODO - math
4059 \ifcase\bbl@select@type % TODO - strictly, not the right test
4060 \bbl@bodydir{#1}%
4061 \bbl@pardir{#1}%
4062 \fi
4063 \bbl@texdir{#1}}
4064 % TODO. Only if \bbl@bidimode > 0?:
4065 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4066 \DisableBabelHook{babel-bidi}

```

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

```

4067 \ifodd\bbl@engine % luatex=1
4068 \else % pdftex=0, xetex=2
4069 \newcount\bbl@dirlevel
4070 \chardef\bbl@thetexdir\z@
4071 \chardef\bbl@thepardir\z@
4072 \def\bbl@texdir#1{%
4073 \ifcase#1\relax
4074 \chardef\bbl@thetexdir\z@
4075 \bbl@texdir@i\beginL\endL
4076 \else
4077 \chardef\bbl@thetexdir\@ne
4078 \bbl@texdir@i\beginR\endR
4079 \fi}
4080 \def\bbl@texdir@i#1#2{%
4081 \ifhmode
4082 \ifnum\currentgrouplevel>\z@
4083 \ifnum\currentgrouplevel=\bbl@dirlevel
4084 \bbl@error{Multiple bidi settings inside a group}%
4085 {I'll insert a new group, but expect wrong results.}%
4086 \bgroup\aftergroup#2\aftergroup\egroup
4087 \else
4088 \ifcase\currentgrouptype\or % 0 bottom
4089 \aftergroup#2% 1 simple {}
4090 \or
4091 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4092 \or
4093 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4094 \or\or\or % vbox vtop align
4095 \or
4096 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4097 \or\or\or\or\or\or % output math disc insert vcent mathchoice
4098 \or
4099 \aftergroup#2% 14 \begingroup
4100 \else
4101 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4102 \fi
4103 \fi
4104 \bbl@dirlevel\currentgrouplevel
4105 \fi
4106 #1%
4107 \fi}
4108 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}

```

```

4109 \let\bbl@bodydir\@gobble
4110 \let\bbl@pagedir\@gobble
4111 \def\bbl@dirparastext{\chardef\bbl@thepardir\bbl@thetextdir}

```

The following command is executed only if there is a right-to-left script (once). It activates the `\everypar` hack for xetex, to properly handle the par direction. Note text and par dirs are decoupled to some extent (although not completely).

```

4112 \def\bbl@xebidipar{%
4113   \let\bbl@xebidipar\relax
4114   \TeXeTstate\@ne
4115   \def\bbl@xeeverypar{%
4116     \ifcase\bbl@thepardir
4117       \ifcase\bbl@thetextdir\else\beginR\fi
4118     \else
4119       {\setbox\z@\lastbox\beginR\box\z@}%
4120     \fi}%
4121   \let\bbl@severypar\everypar
4122   \newtoks\everypar
4123   \everypar=\bbl@severypar
4124   \bbl@severypar{\bbl@xeeverypar\the\everypar}}
4125 \ifnum\bbl@bidimode>200
4126   \let\bbl@textdir@i\@gobbletwo
4127   \let\bbl@xebidipar\@empty
4128   \AddBabelHook{bidi}{foreign}{%
4129     \def\bbl@tempa{\def\BabelText###1}%
4130     \ifcase\bbl@thetextdir
4131       \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4132     \else
4133       \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4134     \fi}
4135   \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4136   \fi
4137 \fi

```

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

```

4138 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4139 \AtBeginDocument{%
4140   \ifx\pdfstringdefDisableCommands\@undefined\else
4141     \ifx\pdfstringdefDisableCommands\relax\else
4142       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4143     \fi
4144   \fi}

```

## 8.6 Local Language Configuration

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

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

```

4145 \bbl@trace{Local Language Configuration}
4146 \ifx\loadlocalcfg\@undefined
4147   \@ifpackagewith{babel}{noconfigs}%
4148   {\let\loadlocalcfg\@gobble}%
4149   {\def\loadlocalcfg#1{%
4150     \InputIfFileExists{#1.cfg}%
4151     {\typeout{*****^J%
4152               * Local config file #1.cfg used^^J%
4153               *}}%
4154     \@empty}}
4155 \fi

```

## 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).

```
4156 \bbl@trace{Language options}
4157 \let\bbl@afterlang\relax
4158 \let\BabelModifiers\relax
4159 \let\bbl@loaded\@empty
4160 \def\bbl@load@language#1{%
4161   \InputIfFileExists{#1.ldf}%
4162   {\edef\bbl@loaded{\CurrentOption
4163     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4164     \expandafter\let\expandafter\bbl@afterlang
4165     \csname\CurrentOption.ldf-h@@k\endcsname
4166     \expandafter\let\expandafter\BabelModifiers
4167     \csname\bbl@mod@\CurrentOption\endcsname}%
4168   {\bbl@error{%
4169     Unknown option '\CurrentOption'. Either you misspelled it\\%
4170     or the language definition file \CurrentOption.ldf was not found}}%
4171     Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4172     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4173     headfoot=, strings=, config=, hyphenmap=, or a language name.}}
```

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

```
4174 \def\bbl@try@load@lang#1#2#3{%
4175   \IfFileExists{\CurrentOption.ldf}%
4176   {\bbl@load@language{\CurrentOption}}%
4177   {#1\bbl@load@language{#2}#3}}
4178 %
4179 \DeclareOption{hebrew}{%
4180   \input{rlbabel.def}%
4181   \bbl@load@language{hebrew}}
4182 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4183 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4184 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4185 \DeclareOption{polutonikogreek}{%
4186   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4187 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4188 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4189 \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.

```
4190 \ifx\bbl@opt@config\@nnil
4191   \@ifpackagewith{babel}{noconfigs}{}%
4192   {\InputIfFileExists{bblopts.cfg}%
4193     {\typeout{*****^J%
4194       * Local config file bblopts.cfg used^^J%
4195       *}}%
4196     {}}%
4197 \else
4198   \InputIfFileExists{\bbl@opt@config.cfg}%
4199   {\typeout{*****^J%
4200     * Local config file \bbl@opt@config.cfg used^^J%
4201     *}}%
4202   {\bbl@error{%
4203     Local config file '\bbl@opt@config.cfg' not found}%
4204     Perhaps you misspelled it.}}%
4205 \fi
```

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

```

4206 \ifx\bbl@opt@main\@nnil
4207   \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4208     \let\bbl@tempb\@empty
4209     \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4210     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4211     \bbl@foreach\bbl@tempb{%      \bbl@tempb is a reversed list
4212       \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4213         \ifodd\bbl@iniflag % = *=
4214           \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
4215         \else % n +=
4216           \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4217         \fi
4218       \fi}%
4219   \fi
4220 \else
4221   \bbl@info{Main language set with 'main='. Except if you have\\%
4222     problems, prefer the default mechanism for setting\\%
4223     the main language. Reported}%
4224 \fi

```

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

```

4225 \ifx\bbl@opt@main\@nnil\else
4226   \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
4227   \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4228 \fi

```

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

```

4229 \bbl@foreach\bbl@language@opts{%
4230   \def\bbl@tempa{#1}%
4231   \ifx\bbl@tempa\bbl@opt@main\else
4232     \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4233       \bbl@ifunset{ds@#1}%
4234       {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4235       {}%
4236     \else % + * (other = ini)
4237       \DeclareOption{#1}{%
4238         \bbl@ldfinit
4239         \babelprovide[import]{#1}%
4240         \bbl@afterldf{}}%
4241     \fi
4242   \fi}%
4243 \bbl@foreach\@classoptionslist{%
4244   \def\bbl@tempa{#1}%
4245   \ifx\bbl@tempa\bbl@opt@main\else
4246     \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4247       \bbl@ifunset{ds@#1}%
4248       {\IfFileExists{#1.ldf}%
4249        {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4250        {}}%
4251     \else % + * (other = ini)
4252       \IfFileExists{babel-#1.tex}%
4253       {\DeclareOption{#1}{%
4254         \bbl@ldfinit

```

```

4256         \babelprovide[import]{#1}%
4257         \bbl@afterldf{}}}%
4258     {}%
4259     \fi
4260 \fi}

```

And we are done, because all options for this pass has been declared. Those already processed in the first pass are just ignored.

The options have to be processed in the order in which the user specified them (but remember class options are processed before):

```

4261 \def\AfterBabelLanguage#1{%
4262   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4263 \DeclareOption*{}
4264 \ProcessOptions*

```

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

```

4265 \bbl@trace{Option 'main'}
4266 \ifx\bbl@opt@main\@nnil
4267   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4268   \let\bbl@tempc\@empty
4269   \bbl@for\bbl@tempb\bbl@tempa{%
4270     \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
4271     \ifin\edef\bbl@tempc{\bbl@tempb}\fi}
4272   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4273   \expandafter\bbl@tempa\bbl@loaded,\@nnil
4274   \ifx\bbl@tempb\bbl@tempc\else
4275     \bbl@warning{%
4276       Last declared language option is '\bbl@tempc',\%
4277       but the last processed one was '\bbl@tempb'.\%
4278       The main language can't be set as both a global\%
4279       and a package option. Use 'main=\bbl@tempc' as\%
4280       option. Reported}
4281   \fi
4282 \else
4283   \ifodd\bbl@iniflag % case 1,3 (main is ini)
4284     \bbl@ldfinit
4285     \let\CurrentOption\bbl@opt@main
4286     \bbl@exp{% \bbl@opt@provide = empty if *
4287       \\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4288     \bbl@afterldf{}
4289     \DeclareOption{\bbl@opt@main}{}
4290   \else % case 0,2 (main is ldf)
4291     \ifx\bbl@loadmain\relax
4292       \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4293     \else
4294       \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4295     \fi
4296     \ExecuteOptions{\bbl@opt@main}
4297     \@namedef{ds@\bbl@opt@main}{}%
4298   \fi
4299   \DeclareOption*{}
4300   \ProcessOptions*
4301 \fi
4302 \def\AfterBabelLanguage{%
4303   \bbl@error
4304   {Too late for \string\AfterBabelLanguage}%
4305   {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.

```
4306 \ifx\bbl@main@language\@undefined
4307   \bbl@info{%
4308     You haven't specified a language. I'll use 'nil'\%
4309     as the main language. Reported}
4310   \bbl@load@language{nil}
4311 \fi
4312 \</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`, `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`

```
4313 \<*kernel>
4314 \let\bbl@onlyswitch\@empty
4315 \input babel.def
4316 \let\bbl@onlyswitch\@undefined
4317 \</kernel>
4318 \<*patterns>
```

## 10 Loading hyphenation patterns

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

```
4319 \<Make sure ProvidesFile is defined>
4320 \ProvidesFile{hyphen.cfg}[\<<date>> \<<version>> Babel hyphens]
4321 \xdef\bbl@format{\jobname}
4322 \def\bbl@version{\<<version>>}
4323 \def\bbl@date{\<<date>>}
4324 \ifx\AtBeginDocument\@undefined
4325   \def\@empty{}
4326 \fi
4327 \<Define core switching macros>
```

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

```
4328 \def\process@line#1#2 #3 #4 {%
4329   \ifx=#1%
4330     \process@synonym{#2}%
4331   \else
4332     \process@language{#1#2}{#3}{#4}%
4333   \fi
4334   \ignorespaces}
```

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

```
4335 \toks@{}
4336 \def\bbl@languages{}
```

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

```

4337 \def\process@synonym#1{%
4338   \ifnum\last@language=\m@ne
4339     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4340   \else
4341     \expandafter\chardef\csname l@#1\endcsname\last@language
4342     \wlog{\string\l@#1=\string\language\the\last@language}%
4343     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4344       \csname\language\hyphenmins\endcsname
4345     \let\bbl@elt\relax
4346     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4347   \fi}

```

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

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

For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file `language.dat` by adding for instance ‘:T1’ to the name of the language.

The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

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

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

Then we globally store the settings of `\lefthyphenmin` and `\righthyphenmin` and close the group.

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

`\bbl@languages` saves a snapshot of the loaded languages in the form

`\bbl@elt{<language-name>}{<number>}{<patterns-file>}{<exceptions-file>}`. Note the last 2

arguments are empty in ‘dialects’ defined in `language.dat` with =. Note also the language name can have encoding info.

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

```

4348 \def\process@language#1#2#3{%
4349   \expandafter\addlanguage\csname l@#1\endcsname
4350   \expandafter\language\csname l@#1\endcsname
4351   \edef\language{#1}%
4352   \bbl@hook@everylanguage{#1}%
4353   % > luatex
4354   \bbl@get@enc#1::\@@@
4355   \begingroup
4356     \lefthyphenmin\m@ne
4357     \bbl@hook@loadpatterns{#2}%
4358     % > luatex
4359     \ifnum\lefthyphenmin=\m@ne
4360     \else
4361       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4362         \the\lefthyphenmin\the\righthyphenmin}%
4363     \fi
4364   \endgroup
4365   \def\bbl@tempa{#3}%
4366   \ifx\bbl@tempa\@empty\else

```



```

4367 \bbl@hook@loadexceptions{#3}%
4368 % > luatex
4369 \fi
4370 \let\bbl@elt\relax
4371 \edef\bbl@languages{%
4372 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4373 \ifnum\the\language=\z@
4374 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4375 \set@hyphenmins\tw@\thr@\relax
4376 \else
4377 \expandafter\expandafter\expandafter\set@hyphenmins
4378 \csname #1hyphenmins\endcsname
4379 \fi
4380 \the\toks@
4381 \toks@{}%
4382 \fi}

```

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

```

4383 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}

```

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

```

4384 \def\bbl@hook@everylanguage#1{}
4385 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4386 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4387 \def\bbl@hook@loadkernel#1{%
4388 \def\addlanguage{\csname newlanguage\endcsname}%
4389 \def\adddialect##1##2{%
4390 \global\chardef##1##2\relax
4391 \wlog{\string##1 = a dialect from \string\language##2}}%
4392 \def\iflanguage##1{%
4393 \expandafter\ifx\csname l@##1\endcsname\relax
4394 \@nolanerr{##1}%
4395 \else
4396 \ifnum\csname l@##1\endcsname=\language
4397 \expandafter\expandafter\expandafter\@firstoftwo
4398 \else
4399 \expandafter\expandafter\expandafter\@secondoftwo
4400 \fi
4401 \fi}%
4402 \def\providehyphenmins##1##2{%
4403 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4404 \@namedef{##1hyphenmins}{##2}%
4405 \fi}%
4406 \def\set@hyphenmins##1##2{%
4407 \lefthyphenmin##1\relax
4408 \righthyphenmin##2\relax}%
4409 \def\selectlanguage{%
4410 \errhelp{Selecting a language requires a package supporting it}%
4411 \errmessage{Not loaded}}%
4412 \let\foreignlanguage\selectlanguage
4413 \let\otherlanguage\selectlanguage
4414 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4415 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4416 \def\setlocale{%
4417 \errhelp{Find an armchair, sit down and wait}%
4418 \errmessage{Not yet available}}%
4419 \let\uselocale\setlocale
4420 \let\locale\setlocale
4421 \let\selectlocale\setlocale
4422 \let\localename\setlocale

```

```

4423 \let\textlocale\setlocale
4424 \let\textlanguage\setlocale
4425 \let\languagetext\setlocale}
4426 \begingroup
4427 \def\AddBabelHook#1#2{%
4428 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4429 \def\next{\toks1}%
4430 \else
4431 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4432 \fi
4433 \next}
4434 \ifx\directlua\undefined
4435 \ifx\XeTeXinputencoding\undefined\else
4436 \input xebabel.def
4437 \fi
4438 \else
4439 \input luababel.def
4440 \fi
4441 \openin1 = babel-\bbl@format.cfg
4442 \ifeof1
4443 \else
4444 \input babel-\bbl@format.cfg\relax
4445 \fi
4446 \closein1
4447 \endgroup
4448 \bbl@hook@loadkernel{switch.def}

```

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

```

4449 \openin1 = language.dat

```

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

```

4450 \def\languagename{english}%
4451 \ifeof1
4452 \message{I couldn't find the file language.dat,\space
4453         I will try the file hyphen.tex}
4454 \input hyphen.tex\relax
4455 \chardef\l@english\z@
4456 \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.

```

4457 \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.

```

4458 \loop
4459 \endlinechar\m@ne
4460 \read1 to \bbl@line
4461 \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.

```

4462 \if T\ifeof1F\fi T\relax
4463 \ifx\bbl@line\empty\else
4464 \edef\bbl@line{\bbl@line\space\space\space}%
4465 \expandafter\process@line\bbl@line\relax
4466 \fi
4467 \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.

```

4468 \begingroup
4469 \def\bbl@elt#1#2#3#4{%
4470   \global\language=#2\relax
4471   \gdef\language#1}%
4472   \def\bbl@elt##1##2##3##4{}}%
4473 \bbl@languages
4474 \endgroup
4475 \fi
4476 \closein1

```

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

```

4477 \if/\the\toks@/\else
4478   \errhelp{language.dat loads no language, only synonyms}
4479   \errmessage{Orphan language synonym}
4480 \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.

```

4481 \let\bbl@line\@undefined
4482 \let\process@line\@undefined
4483 \let\process@synonym\@undefined
4484 \let\process@language\@undefined
4485 \let\bbl@get@enc\@undefined
4486 \let\bbl@hyph@enc\@undefined
4487 \let\bbl@tempa\@undefined
4488 \let\bbl@hook@loadkernel\@undefined
4489 \let\bbl@hook@everylanguage\@undefined
4490 \let\bbl@hook@loadpatterns\@undefined
4491 \let\bbl@hook@loadexceptions\@undefined
4492 \patterns>

```

Here the code for `initTeX` ends.

## 11 Font handling with fontspec

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

```

4493 <<(*More package options)>> ≡
4494 \chardef\bbl@bidimode\z@
4495 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4496 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4497 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4498 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4499 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4500 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4501 <</More package options>>

```

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

At the time of this writing, `fontspec` shows a warning about there are languages not available, which some people think refers to babel, even if there is nothing wrong. Here is hack to patch `fontspec` to avoid the misleading message, which is replaced by a more explanatory one.

```

4502 <<(*Font selection)>> ≡
4503 \bbl@trace{Font handling with fontspec}
4504 \ifx\ExplSyntaxOn\@undefined\else
4505   \def\bbl@fs@warn@nx#1#2{% \bbl@tempfs is the original macro
4506     \in@{, #1, }{, no-script, language-not-exist,}%
4507     \ifin\else\bbl@tempfs@nx{#1}{#2}\fi}

```

```

4508 \def\bbl@fs@warn@nxx#1#2#3{%
4509 \in@{, #1, }{, no-script, language-not-exist, }%
4510 \ifin@else\bbl@tempfs@nxx{#1}{#2}{#3}\fi}
4511 \def\bbl@loadfontspec{%
4512 \let\bbl@loadfontspec\relax
4513 \ifx\fontspec\undefined
4514 \usepackage{fontspec}%
4515 \fi}%
4516 \fi
4517 \@onlypreamble\babelfont
4518 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4519 \bbl@foreach{#1}{%
4520 \expandafter\ifx\csname date##1\endcsname\relax
4521 \IfFileExists{babel-##1.tex}%
4522 {\babelprovide{##1}}%
4523 }%
4524 \fi}%
4525 \edef\bbl@tempa{#1}%
4526 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4527 \bbl@loadfontspec
4528 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4529 \bbl@bblfont}
4530 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4531 \bbl@ifunset{\bbl@tempb family}%
4532 {\bbl@providefam{\bbl@tempb}}%
4533 }%
4534 % For the default font, just in case:
4535 \bbl@ifunset{\bbl@lsys@languagename}{\bbl@provide@lsys{\languagename}}{}%
4536 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4537 {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
4538 \bbl@exp{%
4539 \let<\bbl@\bbl@tempb dflt@\languagename>\<\bbl@\bbl@tempb dflt@>%
4540 \\\bbl@font@set<\bbl@\bbl@tempb dflt@\languagename>%
4541 \<\bbl@tempb default>\<\bbl@tempb family>}}%
4542 {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4543 \bbl@csarg\def{\bbl@tempb dflt@##1}{<>{#1}{#2}}}%

```

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

```

4544 \def\bbl@providefam#1{%
4545 \bbl@exp{%
4546 \\\newcommand<#1default>{}% Just define it
4547 \\\bbl@add@list\\bbl@font@fams{#1}%
4548 \\\DeclareRobustCommand<#1family>{%
4549 \\\not@math@alphabet<#1family>\relax
4550 % \\\prepare@family@series@update{#1}<#1default>% TODO. Fails
4551 \\\fontfamily<#1default>%
4552 <\ifx>\\\UseHooks\\\@undefined<else>\\\UseHook{#1family}<\fi>%
4553 \\\selectfont}%
4554 \\\DeclareTextFontCommand{\<text#1>}{<#1family>}}

```

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

```

4555 \def\bbl@nostdfont#1{%
4556 \bbl@ifunset{\bbl@WFF@\f@family}%
4557 {\bbl@csarg\gdef{WFF@\f@family}}{% Flag, to avoid dupl warns
4558 \bbl@infowarn{The current font is not a babel standard family:\\%
4559 #1%
4560 \fontname\font\\%
4561 There is nothing intrinsically wrong with this warning, and\\%
4562 you can ignore it altogether if you do not need these\\%
4563 families. But if they are used in the document, you should be\\%
4564 aware 'babel' will not set Script and Language for them, so\\%
4565 you may consider defining a new family with \string\babelfont.\\%
4566 See the manual for further details about \string\babelfont.\\%

```

```

4567     Reported}}
4568   {}}%
4569 \gdef\bb@switchfont{%
4570   \bb@ifunset{\bb@lsys@\languagename}{\bb@provide@lsys{\languagename}}{}%
4571   \bb@exp{%   eg Arabic -> arabic
4572     \lowercase{\edef\\bb@tempa{\bb@cl{sname}}}}}%
4573   \bb@foreach\bb@font@fams{%
4574     \bb@ifunset{\bb@##1dflt@\languagename}%      (1) language?
4575     {\bb@ifunset{\bb@##1dflt@*\bb@tempa}%      (2) from script?
4576       {\bb@ifunset{\bb@##1dflt@}%      2=F - (3) from generic?
4577         {}%      123=F - nothing!
4578         {\bb@exp{%      3=T - from generic
4579           \global\let<\bb@##1dflt@\languagename>%
4580             \<\bb@##1dflt@>}}}%
4581         {\bb@exp{%      2=T - from script
4582           \global\let<\bb@##1dflt@\languagename>%
4583             \<\bb@##1dflt@*\bb@tempa>}}}%
4584         {}}%      1=T - language, already defined
4585   \def\bb@tempa{\bb@nostdfont{}}}%
4586   \bb@foreach\bb@font@fams{%      don't gather with prev for
4587     \bb@ifunset{\bb@##1dflt@\languagename}%
4588     {\bb@cs{famrst@##1}%
4589       \global\bb@csarg\let{famrst@##1}\relax}%
4590     {\bb@exp{% order is relevant. TODO: but sometimes wrong!
4591       \\bb@add\\originalTeX%
4592       \\bb@font@rst{\bb@cl{##1dflt}}}%
4593       \<##1default>\<##1family>{##1}}}%
4594       \\bb@font@set<\bb@##1dflt@\languagename>% the main part!
4595       \<##1default>\<##1family>}}}%
4596   \bb@ifrestoring{{\bb@tempa}}}%

```

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

```

4597 \ifx\fb@family\undefined\else   % if latex
4598   \ifcase\bb@engine              % if pdftex
4599     \let\bb@ckeckstdfonts\relax
4600   \else
4601     \def\bb@ckeckstdfonts{%
4602       \begingroup
4603       \global\let\bb@ckeckstdfonts\relax
4604       \let\bb@tempa\@empty
4605       \bb@foreach\bb@font@fams{%
4606         \bb@ifunset{\bb@##1dflt@}%
4607         {\@nameuse{##1family}%
4608           \bb@csarg\gdef{WFF@\fb@family}}}% Flag
4609         \bb@exp{\\bb@add\\bb@tempa{* \<##1family>= \fb@family\\}%
4610           \space\space\fontname\font\\}%
4611         \bb@csarg\xdef{##1dflt@}{\fb@family}%
4612         \expandafter\xdef\csname ##1default\endcsname{\fb@family}}}%
4613       {}}%
4614     \ifx\bb@tempa\@empty\else
4615       \bb@infowarn{The following font families will use the default\\%
4616         settings for all or some languages:\\%
4617         \bb@tempa
4618         There is nothing intrinsically wrong with it, but\\%
4619         'babel' will no set Script and Language, which could\\%
4620         be relevant in some languages. If your document uses\\%
4621         these families, consider redefining them with \string\babelfont.\\%
4622         Reported}%
4623       \fi
4624     \endgroup
4625   \fi
4626 \fi

```

Now the macros defining the font with fontspec.

When there are repeated keys in fontspec, the last value wins. So, we just place the ini settings at the beginning, and user settings will take precedence. We must deactivate temporarily `\bbl@mapselect` because `\selectfont` is called internally when a font is defined.

```

4627 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4628   \bbl@xin@{<>}{#1}%
4629   \ifin@
4630     \bbl@exp{\bbl@fontspec@set\#1\expandafter@gobbletwo#1\#3}%
4631   \fi
4632   \bbl@exp{%
4633     \def\#2{#1}% eg, \rmdefault{\bbl@rmdflt@lang}
4634     \bbl@ifsamestring{#2}{\f@family}%
4635     {\#3%
4636       \bbl@ifsamestring{\f@series}{\bfdefault}{\bfseries}}%
4637     \let\bbl@tempa\relax}%
4638   {}}
4639 % TODO - next should be global?, but even local does its job. I'm
4640 % still not sure -- must investigate:
4641 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4642   \let\bbl@tempe\bbl@mapselect
4643   \let\bbl@mapselect\relax
4644   \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4645   \let#4\empty % Make sure \renewfontfamily is valid
4646   \bbl@exp{%
4647     \let\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
4648     \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4649     {\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4650     \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4651     {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4652     \let\bbl@tempfs@nx<__fontspec_warning:nx>%
4653     \let<__fontspec_warning:nx>\bbl@fs@warn@nx
4654     \let\bbl@tempfs@nxx<__fontspec_warning:nxx>%
4655     \let<__fontspec_warning:nxx>\bbl@fs@warn@nxx
4656     \renewfontfamily\#4%
4657     [\bbl@cl{sys},#2]{#3}% ie \bbl@exp{..}{#3}
4658   \bbl@exp{%
4659     \let<__fontspec_warning:nx>\bbl@tempfs@nx
4660     \let<__fontspec_warning:nxx>\bbl@tempfs@nxx}%
4661   \begingroup
4662     #4%
4663     \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4664   \endgroup
4665   \let#4\bbl@temp@fam
4666   \bbl@exp{\let<\bbl@stripslash#4\space>\bbl@temp@pfam
4667   \let\bbl@mapselect\bbl@tempe}%

```

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

```

4668 \def\bbl@font@rst#1#2#3#4{%
4669   \bbl@csarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}

```

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

```

4670 \def\bbl@font@fams{rm,sf,tt}
4671 <</Font selection>>

```

## 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.

```

4672 <<{*Footnote changes}>> ≡

```

```

4673 \bbl@trace{Bidi footnotes}
4674 \ifnum\bbl@bidimode>\z@
4675   \def\bbl@footnote#1#2#3{%
4676     \@ifnextchar[%
4677       {\bbl@footnote@o{#1}{#2}{#3}}%
4678       {\bbl@footnote@x{#1}{#2}{#3}}}
4679   \long\def\bbl@footnote@x#1#2#3#4{%
4680     \bgroup
4681       \select@language@x{\bbl@main@language}%
4682       \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4683     \egroup}
4684   \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4685     \bgroup
4686       \select@language@x{\bbl@main@language}%
4687       \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4688     \egroup}
4689   \def\bbl@footnotetext#1#2#3{%
4690     \@ifnextchar[%
4691       {\bbl@footnotetext@o{#1}{#2}{#3}}%
4692       {\bbl@footnotetext@x{#1}{#2}{#3}}}
4693   \long\def\bbl@footnotetext@x#1#2#3#4{%
4694     \bgroup
4695       \select@language@x{\bbl@main@language}%
4696       \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4697     \egroup}
4698   \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4699     \bgroup
4700       \select@language@x{\bbl@main@language}%
4701       \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4702     \egroup}
4703   \def\BabelFootnote#1#2#3#4{%
4704     \ifx\bbl@fn@footnote\undefined
4705       \let\bbl@fn@footnote\footnote
4706     \fi
4707     \ifx\bbl@fn@footnotetext\undefined
4708       \let\bbl@fn@footnotetext\footnotetext
4709     \fi
4710     \bbl@ifblank{#2}%
4711     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4712      \@namedef{\bbl@stripslash#1text}%
4713       {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4714     {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4715      \@namedef{\bbl@stripslash#1text}%
4716       {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4717   \fi
4718 <</Footnote changes>>

```

Now, the code.

```

4719 <*xetex>
4720 \def\BabelStringsDefault{unicode}
4721 \let\xebbl@stop\relax
4722 \AddBabelHook{xetex}{encodedcommands}{%
4723   \def\bbl@tempa{#1}%
4724   \ifx\bbl@tempa\empty
4725     \XeTeXinputencoding"bytes"%
4726   \else
4727     \XeTeXinputencoding"#1"%
4728   \fi
4729   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4730 \AddBabelHook{xetex}{stopcommands}{%
4731   \xebbl@stop
4732   \let\xebbl@stop\relax}
4733 \def\bbl@intraspace#1 #2 #3\@@{%

```

```

4734 \bbl@csarg\gdef{\xeisp@\language\language}%
4735 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4736 \def\bbl@intrapenalty#1\@@{%
4737 \bbl@csarg\gdef{\xeipn@\language\language}%
4738 {\XeTeXlinebreakpenalty #1\relax}}
4739 \def\bbl@provide@intraspace{%
4740 \bbl@xin@{/s}{/\bbl@cl{lnbrk}}}%
4741 \ifin@ \else \bbl@xin@{/c}{/\bbl@cl{lnbrk}} \fi
4742 \ifin@
4743 \bbl@ifunset{\bbl@intsp@\language\language}{}%
4744 {\expandafter\ifx\csname\bbl@intsp@\language\language\endcsname\@empty\else
4745 \ifx\bbl@KVP@intraspace\@nnil
4746 \bbl@exp{%
4747 \bbl@intraspace\bbl@cl{intsp}\@@}%
4748 \fi
4749 \ifx\bbl@KVP@intrapenalty\@nnil
4750 \bbl@intrapenalty0\@@
4751 \fi
4752 \fi
4753 \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4754 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4755 \fi
4756 \ifx\bbl@KVP@intrapenalty\@nnil\else
4757 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4758 \fi
4759 \bbl@exp{%
4760 % TODO. Execute only once (but redundant):
4761 \bbl@add\<extras\language\language>{%
4762 \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
4763 \<bbl@xeisp@\language\language>%
4764 \<bbl@xeipn@\language\language>%
4765 \bbl@toglobal\<extras\language\language>%
4766 \bbl@add\<noextras\language\language>{%
4767 \XeTeXlinebreaklocale ""}%
4768 \bbl@toglobal\<noextras\language\language>%
4769 \ifx\bbl@ispacesize\@undefined
4770 \gdef\bbl@ispacesize{\bbl@cl{\xeisp}}}%
4771 \ifx\AtBeginDocument\@notprerr
4772 \expandafter\@secondoftwo % to execute right now
4773 \fi
4774 \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
4775 \fi}%
4776 \fi}
4777 \ifx\DisableBabelHook\@undefined\endinput\fi
4778 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4779 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfonts}
4780 \DisableBabelHook{babel-fontspec}
4781 \langle Font selection \rangle
4782 \input txtbabel.def
4783 \</xetex>

```

## 12.2 Layout

*In progress.*

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titleps, and geometry.

\bbl@startskip and \bbl@endskip are available to package authors. Thanks to the  $\TeX$  expansion mechanism the following constructs are valid: \adim\bbl@startskip, \advance\bbl@startskip\adim, \bbl@startskip\adim.

Consider txtbabel as a shorthand for *tex-xet babel*, which is the bidi model in both pdftex and xetex.

```

4784 \<texxet>
4785 \providecommand\bbl@provide@intraspace{}
4786 \bbl@trace{Redefinitions for bidi layout}

```



```

4787 \def\bbl@sspre@caption{%
4788   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4789 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4790 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4791 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4792 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4793   \def\@hangfrom#1{%
4794     \setbox\@tempboxa\hbox{\{#1\}}%
4795     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4796     \noindent\box\@tempboxa}
4797   \def\raggedright{%
4798     \let\\\@centercr
4799     \bbl@startskip\z@skip
4800     \@rightskip\@flushglue
4801     \bbl@endskip\@rightskip
4802     \parindent\z@
4803     \parfillskip\bbl@startskip}
4804   \def\raggedleft{%
4805     \let\\\@centercr
4806     \bbl@startskip\@flushglue
4807     \bbl@endskip\z@skip
4808     \parindent\z@
4809     \parfillskip\bbl@endskip}
4810 \fi
4811 \IfBabelLayout{lists}
4812 {\bbl@sreplace\list
4813   {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4814   \def\bbl@listleftmargin{%
4815     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4816   \ifcase\bbl@engine
4817     \def\labelenumii{}\theenumii{}% pdftex doesn't reverse ()
4818     \def\p@enumiii{\p@enumii}\theenumii{}%
4819   \fi
4820   \bbl@sreplace\@verbatim
4821     {\leftskip\@totalleftmargin}%
4822     {\bbl@startskip\textwidth
4823       \advance\bbl@startskip-\linewidth}%
4824   \bbl@sreplace\@verbatim
4825     {\rightskip\z@skip}%
4826     {\bbl@endskip\z@skip}}%
4827 {}
4828 \IfBabelLayout{contents}
4829 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4830   \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4831 {}
4832 \IfBabelLayout{columns}
4833 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4834   \def\bbl@outputbox#1{%
4835     \hb@xt@\textwidth{%
4836       \hskip\columnwidth
4837       \hfil
4838       {\normalcolor\vrule \@width\columnseprule}%
4839       \hfil
4840       \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4841       \hskip-\textwidth
4842       \hb@xt@\columnwidth{\box\@outputbox \hss}%
4843       \hskip\columnsep
4844       \hskip\columnwidth}}}%
4845 {}
4846 <Footnote changes>
4847 \IfBabelLayout{footnotes}%
4848   {\BabelFootnote\footnote\language\{}}%
4849   {\BabelFootnote\localfootnote\language\{}}%

```

```

4850 \BabelFootnote\mainfootnote{\}\{\}\}
4851 {}

Implicitly reverses sectioning labels in bidi=basic, because the full stop is not in contact with L
numbers any more. I think there must be a better way.

4852 \IfBabelLayout{counters}%
4853 {\let\bbl@latin@arabic=\@arabic
4854 \def\@arabic#1{\babelsublr{\bbl@latin@arabic#1}}}%
4855 \let\bbl@asci@roman=\@roman
4856 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asci@roman#1}}}%
4857 \let\bbl@asci@Roman=\@Roman
4858 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asci@Roman#1}}}\}
4859 \</texet>

```

## 12.3 LuaTeX

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

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

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

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

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

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

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

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

```

4860 \*luatex
4861 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4862 \bbl@trace{Read language.dat}
4863 \ifx\bbl@readstream\@undefined
4864 \csname newread\endcsname\bbl@readstream
4865 \fi
4866 \begingroup
4867 \toks@{}
4868 \count@% 0=start, 1=0th, 2=normal
4869 \def\bbl@process@line#1#2 #3 #4 {%
4870 \ifx=#1%
4871 \bbl@process@synonym{#2}%
4872 \else
4873 \bbl@process@language{#1#2}{#3}{#4}%
4874 \fi

```

```

4875 \ignorespaces}
4876 \def\bbl@manylang{%
4877 \ifnum\bbl@last>\@ne
4878 \bbl@info{Non-standard hyphenation setup}%
4879 \fi
4880 \let\bbl@manylang\relax}
4881 \def\bbl@process@language#1#2#3{%
4882 \ifcase\count@
4883 \ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4884 \or
4885 \count@\tw@
4886 \fi
4887 \ifnum\count@=\tw@
4888 \expandafter\addlanguage\csname l@#1\endcsname
4889 \language\allocationnumber
4890 \chardef\bbl@last\allocationnumber
4891 \bbl@manylang
4892 \let\bbl@elt\relax
4893 \xdef\bbl@languages{%
4894 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4895 \fi
4896 \the\toks@
4897 \toks@{}}
4898 \def\bbl@process@synonym@aux#1#2{%
4899 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4900 \let\bbl@elt\relax
4901 \xdef\bbl@languages{%
4902 \bbl@languages\bbl@elt{#1}{#2}{}}}%
4903 \def\bbl@process@synonym#1{%
4904 \ifcase\count@
4905 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4906 \or
4907 \ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}}%
4908 \else
4909 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4910 \fi}
4911 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4912 \chardef\l@english\z@
4913 \chardef\l@USenglish\z@
4914 \chardef\bbl@last\z@
4915 \global\@namedef{bbl@hyphendata@0}{\hyphen.tex}}
4916 \gdef\bbl@languages{%
4917 \bbl@elt{english}{0}{\hyphen.tex}}%
4918 \bbl@elt{USenglish}{0}{}}
4919 \else
4920 \global\let\bbl@languages@format\bbl@languages
4921 \def\bbl@elt#1#2#3#4{% Remove all except language 0
4922 \ifnum#2>\z@\else
4923 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4924 \fi}%
4925 \xdef\bbl@languages{\bbl@languages}%
4926 \fi
4927 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4928 \bbl@languages
4929 \openin\bbl@readstream=language.dat
4930 \ifeof\bbl@readstream
4931 \bbl@warning{I couldn't find language.dat. No additional\\%
4932 patterns loaded. Reported}%
4933 \else
4934 \loop
4935 \endlinechar\m@ne
4936 \read\bbl@readstream to \bbl@line
4937 \endlinechar`^^M

```

```

4938 \if T\ifeof\bb1@readstream F\fi T\relax
4939 \ifx\bb1@line\@empty\else
4940 \edef\bb1@line{\bb1@line\space\space\space}%
4941 \expandafter\bb1@process@line\bb1@line\relax
4942 \fi
4943 \repeat
4944 \fi
4945 \endgroup
4946 \bb1@trace{Macros for reading patterns files}
4947 \def\bb1@get@enc#1:#2:#3\@@{\def\bb1@hyph@enc{#2}}
4948 \ifx\babelcatcodetablenum\@undefined
4949 \ifx\newcatcodetable\@undefined
4950 \def\babelcatcodetablenum{5211}
4951 \def\bb1@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4952 \else
4953 \newcatcodetable\babelcatcodetablenum
4954 \newcatcodetable\bb1@pattcodes
4955 \fi
4956 \else
4957 \def\bb1@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4958 \fi
4959 \def\bb1@luapatterns#1#2{%
4960 \bb1@get@enc#1::\@@@
4961 \setbox\z@\hbox\bgroup
4962 \beginingroup
4963 \savecatcodetable\babelcatcodetablenum\relax
4964 \initcatcodetable\bb1@pattcodes\relax
4965 \catcodetable\bb1@pattcodes\relax
4966 \catcode\#=6 \catcode\$_=3 \catcode\&=4 \catcode\^=7
4967 \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\~=13
4968 \catcode\@=11 \catcode\^^I=10 \catcode\^^J=12
4969 \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
4970 \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
4971 \catcode\`=12 \catcode\'=12 \catcode\"=12
4972 \input #1\relax
4973 \catcodetable\babelcatcodetablenum\relax
4974 \endgroup
4975 \def\bb1@tempa{#2}%
4976 \ifx\bb1@tempa\@empty\else
4977 \input #2\relax
4978 \fi
4979 \egroup}%
4980 \def\bb1@patterns@lua#1{%
4981 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4982 \csname l@#1\endcsname
4983 \edef\bb1@tempa{#1}%
4984 \else
4985 \csname l@#1:\f@encoding\endcsname
4986 \edef\bb1@tempa{#1:\f@encoding}%
4987 \fi\relax
4988 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4989 \@ifundefined{bb1@hyphendata@the\language}%
4990 {\def\bb1@elt##1##2##3##4{%
4991 \ifnum##2=\csname l@bb1@tempa\endcsname % #2=spanish, dutch:OT1...
4992 \def\bb1@tempb{##3}%
4993 \ifx\bb1@tempb\@empty\else % if not a synonymous
4994 \def\bb1@tempc{##3}{##4}%
4995 \fi
4996 \bb1@csarg\xdef{hyphendata@##2}{\bb1@tempc}%
4997 \fi}%
4998 \bb1@languages
4999 \@ifundefined{bb1@hyphendata@the\language}%
5000 {\bb1@info{No hyphenation patterns were set for\%

```

```

5001             language '\bbl@tempa'. Reported}}}%
5002         {\expandafter\expandafter\expandafter\bbl@luapatterns
5003             \csname bbl@hyphendata@the\language\endcsname}}}}
5004 \endinput\fi
5005 % Here ends \ifx\AddBabelHook\@undefined
5006 % A few lines are only read by hyphen.cfg
5007 \ifx\DisableBabelHook\@undefined
5008   \AddBabelHook{luatex}{everylanguage}{%
5009     \def\process@language##1##2##3{%
5010       \def\process@line####1####2 ####3 ####4 {}}
5011   \AddBabelHook{luatex}{loadpatterns}{%
5012     \input #1\relax
5013     \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5014       {{#1}}}}
5015   \AddBabelHook{luatex}{loadexceptions}{%
5016     \input #1\relax
5017     \def\bbl@tempb##1##2{{##1}{#1}}%
5018     \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5019       {\expandafter\expandafter\expandafter\bbl@tempb
5020         \csname bbl@hyphendata@the\language\endcsname}}
5021 \endinput\fi
5022 % Here stops reading code for hyphen.cfg
5023 % The following is read the 2nd time it's loaded
5024 \begingroup % TODO - to a lua file
5025 \catcode\%=12
5026 \catcode\`=12
5027 \catcode\`=12
5028 \catcode\:=12
5029 \directlua{
5030   Babel = Babel or {}
5031   function Babel.bytes(line)
5032     return line:gsub(".",
5033       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5034   end
5035   function Babel.begin_process_input()
5036     if luatexbase and luatexbase.add_to_callback then
5037       luatexbase.add_to_callback('process_input_buffer',
5038         Babel.bytes, 'Babel.bytes')
5039     else
5040       Babel.callback = callback.find('process_input_buffer')
5041       callback.register('process_input_buffer', Babel.bytes)
5042     end
5043   end
5044   function Babel.end_process_input ()
5045     if luatexbase and luatexbase.remove_from_callback then
5046       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5047     else
5048       callback.register('process_input_buffer', Babel.callback)
5049     end
5050   end
5051   function Babel.addpatterns(pp, lg)
5052     local lg = lang.new(lg)
5053     local pats = lang.patterns(lg) or ''
5054     lang.clear_patterns(lg)
5055     for p in pp:gmatch('[^s]+') do
5056       ss = ''
5057       for i in string.utfcharacters(p:gsub('%d', '')) do
5058         ss = ss .. '%d?' .. i
5059       end
5060       ss = ss:gsub('^%d%?%', '%%.') .. '%d?'
5061       ss = ss:gsub('%.%d%?$', '%%.')
5062       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5063       if n == 0 then

```

```

5064         tex.sprint(
5065             [[\string\csname\space bbl@info\endcsname{New pattern: }]
5066             .. p .. [{}]])
5067         pats = pats .. ' ' .. p
5068     else
5069         tex.sprint(
5070             [[\string\csname\space bbl@info\endcsname{Renew pattern: }]
5071             .. p .. [{}]])
5072     end
5073 end
5074 lang.patterns(lg, pats)
5075 end
5076 Babel.characters = Babel.characters or {}
5077 Babel.ranges = Babel.ranges or {}
5078 function Babel.hlist_has_bidi(head)
5079     local has_bidi = false
5080     local ranges = Babel.ranges
5081     for item in node.traverse(head) do
5082         if item.id == node.id'glyph' then
5083             local itemchar = item.char
5084             local chardata = Babel.characters[itemchar]
5085             local dir = chardata and chardata.d or nil
5086             if not dir then
5087                 for nn, et in ipairs(ranges) do
5088                     if itemchar < et[1] then
5089                         break
5090                     elseif itemchar <= et[2] then
5091                         dir = et[3]
5092                         break
5093                     end
5094                 end
5095             end
5096             if dir and (dir == 'al' or dir == 'r') then
5097                 has_bidi = true
5098             end
5099         end
5100     end
5101     return has_bidi
5102 end
5103 function Babel.set_chranges_b (script, chrng)
5104     if chrng == '' then return end
5105     texio.write('Replacing ' .. script .. ' script ranges')
5106     Babel.script_blocks[script] = {}
5107     for s, e in string.gmatch(chrng..' ', '(-.)%.%.(-.)%s') do
5108         table.insert(
5109             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5110     end
5111 end
5112 }
5113 \endgroup
5114 \ifx\newattribute\@undefined\else
5115     \newattribute\bbl@attr@locale
5116     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5117     \AddBabelHook{luatex}{beforeextras}{%
5118         \setattribute\bbl@attr@locale\localeid}
5119 \fi
5120 \def\BabelStringsDefault{unicode}
5121 \let\luabbl@stop\relax
5122 \AddBabelHook{luatex}{encodedcommands}{%
5123     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5124     \ifx\bbl@tempa\bbl@tempb\else
5125         \directlua{Babel.begin_process_input()}%
5126     \def\luabbl@stop{%

```

```

5127 \directlua{Babel.end_process_input()}}%
5128 \fi}%
5129 \AddBabelHook{luatex}{stopcommands}{%
5130 \luabbbl@stop
5131 \let\luabbbl@stop\relax}
5132 \AddBabelHook{luatex}{patterns}{%
5133 \@ifundefined{bbl@hyphendata@the\language}%
5134 {\def\bbl@elt##1##2##3##4{%
5135 \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5136 \def\bbl@tempb{##3}%
5137 \ifx\bbl@tempb\@empty\else % if not a synonymous
5138 \def\bbl@tempc{##3}{##4}}%
5139 \fi
5140 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5141 \fi}%
5142 \bbl@languages
5143 \@ifundefined{bbl@hyphendata@the\language}%
5144 {\bbl@info{No hyphenation patterns were set for\%
5145 language '#2'. Reported}}%
5146 {\expandafter\expandafter\expandafter\bbl@luapatterns
5147 \csname bbl@hyphendata@the\language\endcsname}}}%
5148 \@ifundefined{bbl@patterns@}{}%
5149 \begingroup
5150 \bbl@xin@{, \number\language,}{, \bbl@pttnlist}%
5151 \ifin@\else
5152 \ifx\bbl@patterns@\@empty\else
5153 \directlua{ Babel.addpatterns(
5154 [[\bbl@patterns@]], \number\language) }%
5155 \fi
5156 \@ifundefined{bbl@patterns@#1}%
5157 \@empty
5158 {\directlua{ Babel.addpatterns(
5159 [[\space\csname bbl@patterns@#1\endcsname]],
5160 \number\language) }}%
5161 \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5162 \fi
5163 \endgroup}%
5164 \bbl@exp{%
5165 \bbl@ifunset{bbl@prehc@\languagename}{}%
5166 {\bbl@ifblank{\bbl@cs{prehc@\languagename}}{}}%
5167 {\prehyphenchar=\bbl@cl{prehc}\relax}}}%

```

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

```

5168 \@onlypreamble\babelpatterns
5169 \AtEndOfPackage{%
5170 \newcommand\babelpatterns[2][\@empty]{%
5171 \ifx\bbl@patterns@\relax
5172 \let\bbl@patterns@\@empty
5173 \fi
5174 \ifx\bbl@pttnlist\@empty\else
5175 \bbl@warning{%
5176 You must not intermingle \string\selectlanguage\space and\%
5177 \string\babelpatterns\space or some patterns will not\%
5178 be taken into account. Reported}%
5179 \fi
5180 \ifx\@empty#1%
5181 \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5182 \else
5183 \edef\bbl@tempb{\zap@space#1 \@empty}%
5184 \bbl@for\bbl@tempa\bbl@tempb{%
5185 \bbl@fixname\bbl@tempa

```

```

5186 \bbl@iflanguage\bbl@tempa{%
5187 \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5188 \@ifundefined\bbl@patterns@\bbl@tempa}%
5189 \@empty
5190 {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5191 #2}}}%
5192 \fi}}

```

## 12.4 Southeast Asian scripts

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

```

5193 % TODO - to a lua file
5194 \directlua{
5195   Babel = Babel or {}
5196   Babel.linebreaking = Babel.linebreaking or {}
5197   Babel.linebreaking.before = {}
5198   Babel.linebreaking.after = {}
5199   Babel.locale = {} % Free to use, indexed by \localeid
5200   function Babel.linebreaking.add_before(func)
5201     tex.print([[ \noexpand\csname bbl@luahyphenate\endcsname ]])
5202     table.insert(Babel.linebreaking.before, func)
5203   end
5204   function Babel.linebreaking.add_after(func)
5205     tex.print([[ \noexpand\csname bbl@luahyphenate\endcsname ]])
5206     table.insert(Babel.linebreaking.after, func)
5207   end
5208 }
5209 \def\bbl@intraspace#1 #2 #3\@{%
5210   \directlua{
5211     Babel = Babel or {}
5212     Babel.intraspaces = Babel.intraspaces or {}
5213     Babel.intraspaces['\csname bbl@sbc@p\language\endcsname'] = %
5214       {b = #1, p = #2, m = #3}
5215     Babel.locale_props[\the\localeid].intraspace = %
5216       {b = #1, p = #2, m = #3}
5217   }}
5218 \def\bbl@intrapenalty#1\@{%
5219   \directlua{
5220     Babel = Babel or {}
5221     Babel.intrapenalties = Babel.intrapenalties or {}
5222     Babel.intrapenalties['\csname bbl@sbc@p\language\endcsname'] = #1
5223     Babel.locale_props[\the\localeid].intrapenalty = #1
5224   }}
5225 \begingroup
5226 \catcode`\%=12
5227 \catcode`\^=14
5228 \catcode`\'=12
5229 \catcode`\~=12
5230 \gdef\bbl@seaintraspace{^
5231 \let\bbl@seaintraspace\relax
5232 \directlua{
5233   Babel = Babel or {}
5234   Babel.sea_enabled = true
5235   Babel.sea_ranges = Babel.sea_ranges or {}
5236   function Babel.set_chranges (script, chrng)
5237     local c = 0
5238     for s, e in string.gmatch(chrng..' ', '(.-%.(-)%s') do
5239       Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5240       c = c + 1
5241     end

```



```

5242 end
5243 function Babel.sea_disc_to_space (head)
5244     local sea_ranges = Babel.sea_ranges
5245     local last_char = nil
5246     local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5247     for item in node.traverse(head) do
5248         local i = item.id
5249         if i == node.id'glyph' then
5250             last_char = item
5251         elseif i == 7 and item.subtype == 3 and last_char
5252             and last_char.char > 0x0C99 then
5253             quad = font.getfont(last_char.font).size
5254             for lg, rg in pairs(sea_ranges) do
5255                 if last_char.char > rg[1] and last_char.char < rg[2] then
5256                     lg = lg:sub(1, 4) ^% Remove trailing number of, eg, Cyril1
5257                     local intraspace = Babel.intraspaces[lg]
5258                     local intrapenalty = Babel.intrapenalties[lg]
5259                     local n
5260                     if intrapenalty ~= 0 then
5261                         n = node.new(14, 0)      ^% penalty
5262                         n.penalty = intrapenalty
5263                         node.insert_before(head, item, n)
5264                     end
5265                     n = node.new(12, 13)      ^% (glue, spaceskip)
5266                     node.setglue(n, intraspace.b * quad,
5267                                   intraspace.p * quad,
5268                                   intraspace.m * quad)
5269                     node.insert_before(head, item, n)
5270                     node.remove(head, item)
5271                 end
5272             end
5273         end
5274     end
5275 end
5276 }^^
5277 \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.

```

5278 \catcode`\%=14
5279 \gdef\bbl@cjkintraspaces{%
5280     \let\bbl@cjkintraspaces\relax
5281     \directlua{
5282         Babel = Babel or {}
5283         require('babel-data-cjk.lua')
5284         Babel.cjk_enabled = true
5285         function Babel.cjk_linebreak(head)
5286             local GLYPH = node.id'glyph'
5287             local last_char = nil
5288             local quad = 655360      % 10 pt = 655360 = 10 * 65536
5289             local last_class = nil
5290             local last_lang = nil
5291
5292             for item in node.traverse(head) do
5293                 if item.id == GLYPH then
5294
5295                     local lang = item.lang

```

```

5296
5297     local LOCALE = node.get_attribute(item,
5298         Babel.attr_locale)
5299     local props = Babel.locale_props[LOCALE]
5300
5301     local class = Babel.cjk_class[item.char].c
5302
5303     if props.cjk_quotes and props.cjk_quotes[item.char] then
5304         class = props.cjk_quotes[item.char]
5305     end
5306
5307     if class == 'cp' then class = 'cl' end % ]] as CL
5308     if class == 'id' then class = 'I' end
5309
5310     local br = 0
5311     if class and last_class and Babel.cjk_breaks[last_class][class] then
5312         br = Babel.cjk_breaks[last_class][class]
5313     end
5314
5315     if br == 1 and props.linebreak == 'c' and
5316         lang ~= \the\l@nohyphenation\space and
5317         last_lang ~= \the\l@nohyphenation then
5318         local intrapenalty = props.intrapenalty
5319         if intrapenalty ~= 0 then
5320             local n = node.new(14, 0) % penalty
5321             n.penalty = intrapenalty
5322             node.insert_before(head, item, n)
5323         end
5324         local intraspace = props.intraspace
5325         local n = node.new(12, 13) % (glue, spaceskip)
5326         node.setglue(n, intraspace.b * quad,
5327             intraspace.p * quad,
5328             intraspace.m * quad)
5329         node.insert_before(head, item, n)
5330     end
5331
5332     if font.getfont(item.font) then
5333         quad = font.getfont(item.font).size
5334     end
5335     last_class = class
5336     last_lang = lang
5337 else % if penalty, glue or anything else
5338     last_class = nil
5339 end
5340 end
5341 lang.hyphenate(head)
5342 end
5343 }%
5344 \bbl@luahyphenate}
5345 \gdef\bbl@luahyphenate{%
5346 \let\bbl@luahyphenate\relax
5347 \directlua{
5348     luatexbase.add_to_callback('hyphenate',
5349     function (head, tail)
5350         if Babel.linebreaking.before then
5351             for k, func in ipairs(Babel.linebreaking.before) do
5352                 func(head)
5353             end
5354         end
5355         if Babel.cjk_enabled then
5356             Babel.cjk_linebreak(head)
5357         end
5358         lang.hyphenate(head)

```

```

5359     if Babel.linebreaking.after then
5360       for k, func in ipairs(Babel.linebreaking.after) do
5361         func(head)
5362       end
5363     end
5364     if Babel.sea_enabled then
5365       Babel.sea_disc_to_space(head)
5366     end
5367   end,
5368   'Babel.hyphenate')
5369 }
5370 }
5371 \endgroup
5372 \def\bbl@provide@intraspace{%
5373   \bbl@ifunset{\bbl@intsp@{language}}{%
5374     {\expandafter\ifx\csname\bbl@intsp@{language}\endcsname\@empty\else
5375       \bbl@xin@{/c}{\bbl@cl{lnbrk}}}%
5376     \ifin@           % cjk
5377       \bbl@cjk@intraspace
5378       \directlua{
5379         Babel = Babel or {}
5380         Babel.locale_props = Babel.locale_props or {}
5381         Babel.locale_props[\the\localeid].linebreak = 'c'
5382       }%
5383       \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@%
5384       \ifx\bbl@KVP@intrapenalty\@nnil
5385         \bbl@intrapenalty0\@@
5386       \fi
5387     \else           % sea
5388       \bbl@sea@intraspace
5389       \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@%
5390       \directlua{
5391         Babel = Babel or {}
5392         Babel.sea_ranges = Babel.sea_ranges or {}
5393         Babel.set_chranges('\bbl@cl{sbcpr}',
5394           '\bbl@cl{chrng}')
5395       }%
5396       \ifx\bbl@KVP@intrapenalty\@nnil
5397         \bbl@intrapenalty0\@@
5398       \fi
5399     \fi
5400   \fi
5401   \ifx\bbl@KVP@intrapenalty\@nnil\else
5402     \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5403   \fi}}

```

## 12.6 Arabic justification

```

5404 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5405 \def\bblar@chars{%
5406   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5407   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5408   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5409 \def\bblar@elongated{%
5410   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5411   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5412   0649,064A}
5413 \begingroup
5414   \catcode`\_ =11 \catcode`\:=11
5415   \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5416 \endgroup
5417 \gdef\bbl@arabicjust{%
5418   \let\bbl@arabicjust\relax

```

```

5419 \newattribute\bblar@kashida
5420 \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5421 \bblar@kashida=\z@
5422 \bbl@patchfont{\bbl@parsejalt}}%
5423 \directlua{
5424   Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5425   Babel.arabic.elong_map[\the\localeid] = {}
5426   luatexbase.add_to_callback('post_linebreak_filter',
5427     Babel.arabic.justify, 'Babel.arabic.justify')
5428   luatexbase.add_to_callback('hpack_filter',
5429     Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5430 }}%
5431 % Save both node lists to make replacement. TODO. Save also widths to
5432 % make computations
5433 \def\bblar@fetchjalt#1#2#3#4{%
5434   \bbl@exp{\bbl@foreach{#1}}{%
5435     \bbl@ifunset{bblar@JE@##1}%
5436     {\setbox\z@\hbox{^^^^200d\char"##1#2}}%
5437     {\setbox\z@\hbox{^^^^200d\char"@nameuse{bblar@JE@##1}#2}}%
5438   \directlua{%
5439     local last = nil
5440     for item in node.traverse(tex.box[0].head) do
5441       if item.id == node.id'glyph' and item.char > 0x600 and
5442         not (item.char == 0x200D) then
5443         last = item
5444       end
5445     end
5446     Babel.arabic.#3['##1#4'] = last.char
5447   }}%
5448 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5449 % perhaps other tables (falt?, csw?). What about kaf? And diacritic
5450 % positioning?
5451 \gdef\bbl@parsejalt{%
5452   \ifx\addfontfeature\undefined\else
5453     \bbl@xin@{/e}{/\bbl@c1{\lnbrk}}%
5454     \ifin@
5455       \directlua{%
5456         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5457           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5458           tex.print([\string\csname\space bbl@parsejalti\endcsname])
5459         end
5460       }%
5461     \fi
5462   \fi}
5463 \gdef\bbl@parsejalti{%
5464   \begingroup
5465     \let\bbl@parsejalt\relax % To avoid infinite loop
5466     \edef\bbl@tempb{\fontid\font}%
5467     \bblar@nofswarn
5468     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5469     \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5470     \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5471     \addfontfeature{RawFeature+=jalt}%
5472     % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5473     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5474     \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5475     \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5476     \directlua{%
5477       for k, v in pairs(Babel.arabic.from) do
5478         if Babel.arabic.dest[k] and
5479           not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5480           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5481             [Babel.arabic.from[k]] = Babel.arabic.dest[k]

```

```

5482         end
5483     end
5484 }%
5485 \endgroup}
5486 %
5487 \begingroup
5488 \catcode`\#=11
5489 \catcode`\~ =11
5490 \directlua{
5491
5492 Babel.arabic = Babel.arabic or {}
5493 Babel.arabic.from = {}
5494 Babel.arabic.dest = {}
5495 Babel.arabic.justify_factor = 0.95
5496 Babel.arabic.justify_enabled = true
5497
5498 function Babel.arabic.justify(head)
5499   if not Babel.arabic.justify_enabled then return head end
5500   for line in node.traverse_id(node.id'hlist', head) do
5501     Babel.arabic.justify_hlist(head, line)
5502   end
5503   return head
5504 end
5505
5506 function Babel.arabic.justify_hbox(head, gc, size, pack)
5507   local has_inf = false
5508   if Babel.arabic.justify_enabled and pack == 'exactly' then
5509     for n in node.traverse_id(12, head) do
5510       if n.stretch_order > 0 then has_inf = true end
5511     end
5512     if not has_inf then
5513       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5514     end
5515   end
5516   return head
5517 end
5518
5519 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5520   local d, new
5521   local k_list, k_item, pos_inline
5522   local width, width_new, full, k_curr, wt_pos, goal, shift
5523   local subst_done = false
5524   local elong_map = Babel.arabic.elong_map
5525   local last_line
5526   local GLYPH = node.id'glyph'
5527   local KASHIDA = Babel.attr_kashida
5528   local LOCALE = Babel.attr_locale
5529
5530   if line == nil then
5531     line = {}
5532     line.glue_sign = 1
5533     line.glue_order = 0
5534     line.head = head
5535     line.shift = 0
5536     line.width = size
5537   end
5538
5539   % Exclude last line. todo. But-- it discards one-word lines, too!
5540   % ? Look for glue = 12:15
5541   if (line.glue_sign == 1 and line.glue_order == 0) then
5542     elongs = {}      % Stores elongated candidates of each line
5543     k_list = {}      % And all letters with kashida
5544     pos_inline = 0   % Not yet used

```

```

5545
5546 for n in node.traverse_id(GLYPH, line.head) do
5547     pos_inline = pos_inline + 1 % To find where it is. Not used.
5548
5549     % Elongated glyphs
5550     if elong_map then
5551         local locale = node.get_attribute(n, LOCALE)
5552         if elong_map[locale] and elong_map[locale][n.font] and
5553             elong_map[locale][n.font][n.char] then
5554             table.insert(elongs, {node = n, locale = locale} )
5555             node.set_attribute(n.prev, KASHIDA, 0)
5556         end
5557     end
5558
5559     % Tatwil
5560     if Babel.kashida_wts then
5561         local k_wt = node.get_attribute(n, KASHIDA)
5562         if k_wt > 0 then % todo. parameter for multi inserts
5563             table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5564         end
5565     end
5566
5567 end % of node.traverse_id
5568
5569 if #elongs == 0 and #k_list == 0 then goto next_line end
5570 full = line.width
5571 shift = line.shift
5572 goal = full * Babel.arabic.justify_factor % A bit crude
5573 width = node.dimensions(line.head) % The 'natural' width
5574
5575 % == Elongated ==
5576 % Original idea taken from 'chickenize'
5577 while (#elongs > 0 and width < goal) do
5578     subst_done = true
5579     local x = #elongs
5580     local curr = elongs[x].node
5581     local oldchar = curr.char
5582     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5583     width = node.dimensions(line.head) % Check if the line is too wide
5584     % Substitute back if the line would be too wide and break:
5585     if width > goal then
5586         curr.char = oldchar
5587         break
5588     end
5589     % If continue, pop the just substituted node from the list:
5590     table.remove(elongs, x)
5591 end
5592
5593 % == Tatwil ==
5594 if #k_list == 0 then goto next_line end
5595
5596 width = node.dimensions(line.head) % The 'natural' width
5597 k_curr = #k_list
5598 wt_pos = 1
5599
5600 while width < goal do
5601     subst_done = true
5602     k_item = k_list[k_curr].node
5603     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5604         d = node.copy(k_item)
5605         d.char = 0x0640
5606         line.head, new = node.insert_after(line.head, k_item, d)
5607         width_new = node.dimensions(line.head)

```

```

5608         if width > goal or width == width_new then
5609             node.remove(line.head, new) % Better compute before
5610             break
5611         end
5612         width = width_new
5613     end
5614     if k_curr == 1 then
5615         k_curr = #k_list
5616         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5617     else
5618         k_curr = k_curr - 1
5619     end
5620 end
5621
5622 ::next_line::
5623
5624 % Must take into account marks and ins, see luatex manual.
5625 % Have to be executed only if there are changes. Investigate
5626 % what's going on exactly.
5627 if subst_done and not gc then
5628     d = node.hpack(line.head, full, 'exactly')
5629     d.shift = shift
5630     node.insert_before(head, line, d)
5631     node.remove(head, line)
5632 end
5633 end % if process line
5634 end
5635 }
5636 \endgroup
5637 \fi\fi % Arabic just block

```

## 12.7 Common stuff

```

5638 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5639 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ccheckstdfonts}
5640 \DisableBabelHook{babel-fontspec}
5641 <<Font selection>>

```

## 12.8 Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a short function which just traverse the node list to carry out the replacements. The table `loc_to_scr` gets the locale from a script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the `\language` and the `\localeid` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with `/` maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```

5642 % TODO - to a lua file
5643 \directlua{
5644 Babel.script_blocks = {
5645   ['dflt'] = {},
5646   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5647               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5648   ['Armn'] = {{0x0530, 0x058F}},
5649   ['Beng'] = {{0x0980, 0x09FF}},
5650   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5651   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5652   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5653               {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5654   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5655   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5656               {0xAB00, 0xAB2F}},
5657   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5658   % Don't follow strictly Unicode, which places some Coptic letters in

```

```

5659 % the 'Greek and Coptic' block
5660 ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5661 ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5662             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5663             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5664             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5665             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5666             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5667 ['Hebr'] = {{0x0590, 0x05FF}},
5668 ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5669             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5670 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5671 ['Knda'] = {{0x0C80, 0x0CFF}},
5672 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5673             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5674             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5675 ['Lao'] = {{0x0E80, 0x0EFF}},
5676 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5677             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5678             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5679 ['Mahj'] = {{0x11150, 0x1117F}},
5680 ['Mlym'] = {{0x0D00, 0x0D7F}},
5681 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5682 ['Orya'] = {{0x0B00, 0x0B7F}},
5683 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5684 ['Syr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5685 ['Taml'] = {{0x0B80, 0x0BFF}},
5686 ['Telu'] = {{0x0C00, 0x0C7F}},
5687 ['Tfng'] = {{0x2D30, 0x2D7F}},
5688 ['Thai'] = {{0x0E00, 0x0E7F}},
5689 ['Tibt'] = {{0x0F00, 0x0FFF}},
5690 ['Vaii'] = {{0xA500, 0xA63F}},
5691 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5692 }
5693
5694 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5695 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5696 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5697
5698 function Babel.locale_map(head)
5699   if not Babel.locale_mapped then return head end
5700
5701   local LOCALE = Babel.attr_locale
5702   local GLYPH = node.id('glyph')
5703   local inmath = false
5704   local toloc_save
5705   for item in node.traverse(head) do
5706     local toloc
5707     if not inmath and item.id == GLYPH then
5708       % Optimization: build a table with the chars found
5709       if Babel.chr_to_loc[item.char] then
5710         toloc = Babel.chr_to_loc[item.char]
5711       else
5712         for lc, maps in pairs(Babel.loc_to_scr) do
5713           for _, rg in pairs(maps) do
5714             if item.char >= rg[1] and item.char <= rg[2] then
5715               Babel.chr_to_loc[item.char] = lc
5716               toloc = lc
5717               break
5718             end
5719           end
5720         end
5721       end
5722     end

```



```

5722 % Now, take action, but treat composite chars in a different
5723 % fashion, because they 'inherit' the previous locale. Not yet
5724 % optimized.
5725 if not toloc and
5726     (item.char >= 0x0300 and item.char <= 0x036F) or
5727     (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5728     (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5729     toloc = toloc_save
5730 end
5731 if toloc and Babel.locale_props[toloc] and
5732     Babel.locale_props[toloc].letters and
5733     tex.getcatcode(item.char) \string~= 11 then
5734     toloc = nil
5735 end
5736 if toloc and toloc > -1 then
5737     if Babel.locale_props[toloc].lg then
5738         item.lang = Babel.locale_props[toloc].lg
5739         node.set_attribute(item, LOCALE, toloc)
5740     end
5741     if Babel.locale_props[toloc]['/'..item.font] then
5742         item.font = Babel.locale_props[toloc]['/'..item.font]
5743     end
5744     toloc_save = toloc
5745 end
5746 elseif not inmath and item.id == 7 then % Apply recursively
5747     item.replace = item.replace and Babel.locale_map(item.replace)
5748     item.pre      = item.pre and Babel.locale_map(item.pre)
5749     item.post     = item.post and Babel.locale_map(item.post)
5750 elseif item.id == node.id'math' then
5751     inmath = (item.subtype == 0)
5752 end
5753 end
5754 return head
5755 end
5756 }

```

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

```

5757 \newcommand\babelcharproperty[1]{%
5758   \count@=#1\relax
5759   \ifvmode
5760     \expandafter\bbl@chprop
5761   \else
5762     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5763               vertical mode (preamble or between paragraphs)}%
5764     {See the manual for futher info}%
5765   \fi}
5766 \newcommand\bbl@chprop[3][\the\count@]{%
5767   \@tempcnta=#1\relax
5768   \bbl@ifunset{\bbl@chprop@#2}%
5769   {\bbl@error{No property named '#2'. Allowed values are\\%
5770             direction (bc), mirror (bmg), and linebreak (lb)}%
5771    {See the manual for futher info}}%
5772   {}%
5773   \loop
5774     \bbl@cs{chprop@#2}{#3}%
5775     \ifnum\count@<\@tempcnta
5776       \advance\count@\@ne
5777     \repeat}
5778 \def\bbl@chprop@direction#1{%
5779   \directlua{
5780     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5781     Babel.characters[\the\count@]['d'] = '#1'

```

```

5782 }}
5783 \let\bbl@chprop@bc\bbl@chprop@direction
5784 \def\bbl@chprop@mirror#1{%
5785   \directlua{
5786     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5787     Babel.characters[\the\count@]['m'] = '\number#1'
5788   }}
5789 \let\bbl@chprop@bmg\bbl@chprop@mirror
5790 \def\bbl@chprop@linebreak#1{%
5791   \directlua{
5792     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5793     Babel.cjk_characters[\the\count@]['c'] = '#1'
5794   }}
5795 \let\bbl@chprop@lb\bbl@chprop@linebreak
5796 \def\bbl@chprop@locale#1{%
5797   \directlua{
5798     Babel.chr_to_loc = Babel.chr_to_loc or {}
5799     Babel.chr_to_loc[\the\count@] =
5800       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
5801   }}

```

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.

```

5802 \directlua{
5803   Babel.nohyphenation = \the\l@nohyphenation
5804 }

```

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

```

5805 \begingroup
5806 \catcode\~ = 12
5807 \catcode\% = 12
5808 \catcode\& = 14
5809 \catcode\| = 12
5810 \gdef\babelprehyphenation{%&
5811   \@ifnextchar[\bbl@settransform{0}]{\bbl@settransform{0}[]}}
5812 \gdef\babelposthyphenation{%&
5813   \@ifnextchar[\bbl@settransform{1}]{\bbl@settransform{1}[]}}
5814 \gdef\bbl@postlinebreak{\bbl@settransform{2}[]} &% WIP
5815 \gdef\bbl@settransform#1[#2]#3#4#5{%&
5816   \ifcase#1
5817     \bbl@activateprehyphen
5818   \or
5819     \bbl@activateposthyphen
5820   \fi
5821   \begingroup
5822     \def\babeltempa{\bbl@add@list\babeltempb}%&
5823     \let\babeltempb\empty
5824     \def\bbl@tempa{#5}%&
5825     \bbl@replace\bbl@tempa{,}{,}%& TODO. Ugly trick to preserve {}
5826     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{%&
5827       \bbl@ifsamestring{##1}{remove}%&
5828       {\bbl@add@list\babeltempb{nil}}}%&
5829     {\directlua{
5830       local rep = {[#1]}=
5831       rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5832       rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')

```

```

5833     rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5834     if #1 == 0 or #1 == 2 then
5835         rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5836             'space = {' .. '%2, %3, %4' .. '}')
5837         rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5838             'spacefactor = {' .. '%2, %3, %4' .. '}')
5839         rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
5840     else
5841         rep = rep:gsub('(no)%s*=%s*([^\s,]*)', Babel.capture_func)
5842         rep = rep:gsub('(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5843         rep = rep:gsub('(post)%s*=%s*([^\s,]*)', Babel.capture_func)
5844     end
5845     tex.print([[\\string\\babeltempa{}}] .. rep .. [{}]])
5846 }}&%
5847 \\bbl@foreach\\babeltempb{&%
5848     \\bbl@forkv{##1}{&%
5849         \\in{,####1,}{,nil,step,data,remove,insert,string,no,pre,&%
5850             no,post,penalty,kashida,space,spacefactor,&%
5851         \\ifin@\\else
5852             \\bbl@error
5853             {Bad option '####1' in a transform.\\&%
5854                 I'll ignore it but expect more errors}&%
5855             {See the manual for further info.}&%
5856         \\fi}&%
5857 \\let\\bbl@kv@attribute\\relax
5858 \\let\\bbl@kv@label\\relax
5859 \\bbl@forkv{#2}{\\bbl@csarg\\edef{kv##1}{##2}}&%
5860 \\ifx\\bbl@kv@attribute\\relax\\else
5861     \\edef\\bbl@kv@attribute{\\expandafter\\bbl@stripslash\\bbl@kv@attribute}&%
5862 \\fi
5863 \\directlua{
5864     local lbrk = Babel.linebreaking.replacements[#1]
5865     local u = unicode.utf8
5866     local id, attr, label
5867     if #1 == 0 or #1 == 2 then
5868         id = \\the\\csname bbl{id@@#3\\endcsname\\space
5869     else
5870         id = \\the\\csname l@#3\\endcsname\\space
5871     end
5872     \\ifx\\bbl@kv@attribute\\relax
5873         attr = -1
5874     \\else
5875         attr = luatexbase.registernumber'\\bbl@kv@attribute'
5876     \\fi
5877     \\ifx\\bbl@kv@label\\relax\\else &% Same refs:
5878         label = [==[\\bbl@kv@label]==]
5879     \\fi
5880     &% Convert pattern:
5881     local patt = string.gsub([==[#4]==], '%s', '')
5882     if #1 == 0 or #1 == 2 then
5883         patt = string.gsub(patt, '|', ' ')
5884     end
5885     if not u.find(patt, '()', nil, true) then
5886         patt = '()' .. patt .. '()'
5887     end
5888     if #1 == 1 then
5889         patt = string.gsub(patt, '%(%)^', '^()')
5890         patt = string.gsub(patt, '%$(%)', '()$')
5891     end
5892     patt = u.gsub(patt, '{(.)}',
5893         function (n)
5894             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5895         end)

```

```

5896     patt = u.gsub(patt, '{(%x%x%x%x+)}',
5897     function (n)
5898         return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
5899     end)
5900     lbkr[id] = lbkr[id] or {}
5901     table.insert(lbkr[id],
5902     { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
5903 }&%
5904 \endgroup}
5905 \endgroup
5906 \def\bbl@activateposthyphen{%
5907 \let\bbl@activateposthyphen\relax
5908 \directlua{
5909     require('babel-transforms.lua')
5910     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5911 }}
5912 \def\bbl@activateprehyphen{%
5913 \let\bbl@activateprehyphen\relax
5914 \directlua{
5915     require('babel-transforms.lua')
5916     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5917 }}

```

## 12.9 Bidi

As a first step, add a handler for bidi and digits (and potentially other processes) just before luaotfload is applied, which is loaded by default by L<sup>A</sup>T<sub>E</sub>X. Just in case, consider the possibility it has not been loaded.

```

5918 \def\bbl@activate@preotf{%
5919 \let\bbl@activate@preotf\relax % only once
5920 \directlua{
5921     Babel = Babel or {}
5922     %
5923     function Babel.pre_otfload_v(head)
5924         if Babel.numbers and Babel.digits_mapped then
5925             head = Babel.numbers(head)
5926         end
5927         if Babel.bidi_enabled then
5928             head = Babel.bidi(head, false, dir)
5929         end
5930         return head
5931     end
5932     %
5933     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
5934         if Babel.numbers and Babel.digits_mapped then
5935             head = Babel.numbers(head)
5936         end
5937         if Babel.bidi_enabled then
5938             head = Babel.bidi(head, false, dir)
5939         end
5940         return head
5941     end
5942     %
5943     luatexbase.add_to_callback('pre_linebreak_filter',
5944     Babel.pre_otfload_v,
5945     'Babel.pre_otfload_v',
5946     luatexbase.priority_in_callback('pre_linebreak_filter',
5947     'luaotfload.node_processor') or nil)
5948     %
5949     luatexbase.add_to_callback('hpack_filter',
5950     Babel.pre_otfload_h,
5951     'Babel.pre_otfload_h',
5952     luatexbase.priority_in_callback('hpack_filter',

```

```

5953         'luaotfload.node_processor') or nil)
5954     }}

```

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

```

5955 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5956   \let\bbl@beforeforeign\leavevmode
5957   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5958   \RequirePackage{luatexbase}
5959   \bbl@activate@preotf
5960   \directlua{
5961     require('babel-data-bidi.lua')
5962     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
5963       require('babel-bidi-basic.lua')
5964     \or
5965       require('babel-bidi-basic-r.lua')
5966     \fi}
5967   % TODO - to locale_props, not as separate attribute
5968   \newattribute\bbl@attr@dir
5969   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
5970   % TODO. I don't like it, hackish:
5971   \bbl@exp{\output{\bodydir\pagedir\the\output}}
5972   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5973 \fi\fi
5974 \chardef\bbl@thetextdir\z@
5975 \chardef\bbl@thepardir\z@
5976 \def\bbl@getluadir#1{%
5977   \directlua{
5978     if tex.#1dir == 'TLT' then
5979       tex.sprint('0')
5980     elseif tex.#1dir == 'TRT' then
5981       tex.sprint('1')
5982     end}}
5983 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
5984   \ifcase#3\relax
5985     \ifcase\bbl@getluadir{#1}\relax\else
5986       #2 TLT\relax
5987     \fi
5988   \else
5989     \ifcase\bbl@getluadir{#1}\relax
5990       #2 TRT\relax
5991     \fi
5992   \fi}
5993 \def\bbl@thedir{0}
5994 \def\bbl@textdir#1{%
5995   \bbl@setluadir{text}\textdir{#1}%
5996   \chardef\bbl@thetextdir#1\relax
5997   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*3+#1}%
5998   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
5999 \def\bbl@pardir#1{%
6000   \bbl@setluadir{par}\pardir{#1}%
6001   \chardef\bbl@thepardir#1\relax}
6002 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
6003 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
6004 \def\bbl@dirparastext{\pardir\the\textdir\relax}%   %%%
6005 %
6006 \ifnum\bbl@bidimode>\z@
6007   \def\bbl@insidemath{0}%
6008   \def\bbl@everymath{\def\bbl@insidemath{1}}
6009   \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6010   \frozen@everymath\expandafter{%
6011     \expandafter\bbl@everymath\the\def\bbl@insidemath{2}}

```

```

6012 \frozen@everydisplay\expandafter{%
6013 \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6014 \AtBeginDocument{
6015 \directlua{
6016     function Babel.math_box_dir(head)
6017         if not (token.get_macro('bbl@insidemath') == '0') then
6018             if Babel.hlist_has_bidi(head) then
6019                 local d = node.new(node.id'dir')
6020                 d.dir = '+TRT'
6021                 node.insert_before(head, node.has_glyph(head), d)
6022                 for item in node.traverse(head) do
6023                     node.set_attribute(item,
6024                         Babel.attr_dir, token.get_macro('bbl@thedir'))
6025                 end
6026             end
6027         end
6028         return head
6029     end
6030     luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6031         "Babel.math_box_dir", 0)
6032 }}%
6033 \fi

```

## 12.10 Layout

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

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

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

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, `tabular` seems to work (at least in simple cases) with `array`, `tabularx`, `hhline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```

6034 \bbl@trace{Redefinitions for bidi layout}
6035 %
6036 <<{*More package options}>> ≡
6037 \chardef\bbl@eqnpos\z@
6038 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6039 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6040 <</More package options>>
6041 %
6042 \def\BabelNoAMSMath{\let\bbl@noamsmath\relax}
6043 \ifnum\bbl@bidimode>\z@
6044     \ifx\matheqdirmode\@undefined\else
6045         \matheqdirmode\@ne
6046     \fi
6047     \let\bbl@eqnodir\relax
6048     \def\bbl@eqdel{()}
6049     \def\bbl@eqnum{%
6050         {\normalfont\normalcolor
6051         \expandafter\@firstoftwo\bbl@eqdel
6052         \theequation
6053         \expandafter\@secondoftwo\bbl@eqdel}}
6054     \def\bbl@puteqno#1{\eqno\hbox{#1}}
6055     \def\bbl@putleqno#1{\leqno\hbox{#1}}
6056     \def\bbl@eqno@flip#1{%
6057         \ifdim\predisplaysize=-\maxdimen
6058             \eqno
6059             \hb@xt@.01pt{\hb@xt@\displaywidth{\hss{#1}}\hss}%

```

```

6060 \else
6061 \leqno\hbox{#1}%
6062 \fi}
6063 \def\bb1@leqno@flip#1{%
6064 \ifdim\predisplaywidth=-\maxdimen
6065 \leqno
6066 \hb@xt@.01pt{\hss\hb@xt@\displaywidth{#1}\hss}}%
6067 \else
6068 \eqno\hbox{#1}%
6069 \fi}
6070 \AtBeginDocument{%
6071 \ifx\maketag@@@ \@undefined % Normal equation, eqnarray
6072 \AddToHook{env/equation/begin}{%
6073 \ifnum\bb1@thetextdir>\z@
6074 \let\@eqnnum\bb1@eqnum
6075 \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6076 \chardef\bb1@thetextdir\z@
6077 \bb1@add\normalfont{\bb1@eqnodir}%
6078 \ifcase\bb1@eqnpos
6079 \let\bb1@puteqno\bb1@eqno@flip
6080 \or
6081 \let\bb1@puteqno\bb1@leqno@flip
6082 \fi
6083 \fi}%
6084 \ifnum\bb1@eqnpos=\tw@ \else
6085 \def\endequation{\bb1@puteqno{\@eqnnum}$$\@ignoretrue}%
6086 \fi
6087 \AddToHook{env/eqnarray/begin}{%
6088 \ifnum\bb1@thetextdir>\z@
6089 \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6090 \chardef\bb1@thetextdir\z@
6091 \bb1@add\normalfont{\bb1@eqnodir}%
6092 \ifnum\bb1@eqnpos=\@ne
6093 \def\@eqnnum{%
6094 \setbox\z@\hbox{\bb1@eqnum}%
6095 \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6096 \else
6097 \let\@eqnnum\bb1@eqnum
6098 \fi
6099 \fi}
6100 % Hack. YA luatex bug?:
6101 \expandafter\bb1@sreplace\csname] \endcsname{${$}{\eqno\kern.001pt$}$}%
6102 \else % amstex
6103 \ifx\bb1@noamsmath \@undefined
6104 \ifnum\bb1@eqnpos=\@ne
6105 \let\bb1@ams@lap\hbox
6106 \else
6107 \let\bb1@ams@lap\llap
6108 \fi
6109 \ExplSyntaxOn
6110 \bb1@sreplace\intertext@{\normalbaselines}%
6111 {\normalbaselines
6112 \ifx\bb1@eqnodir\relax\else\bb1@paddir\@ne\bb1@eqnodir\fi}%
6113 \ExplSyntaxOff
6114 \def\bb1@ams@tagbox#1#2{#1{\bb1@eqnodir#2}}% #1=hbox|@lap|flip
6115 \ifx\bb1@ams@lap\hbox % leqno
6116 \def\bb1@ams@flip#1{%
6117 \hbox to 0.01pt{\hss\hbox to\displaywidth{#1}\hss}}%
6118 \else % eqno
6119 \def\bb1@ams@flip#1{%
6120 \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}}\hss}}%
6121 \fi
6122 \def\bb1@ams@preset#1{%

```

```

6123         \ifnum\bb1@thetextdir>\z@
6124         \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6125         \bb1@sreplace\textdef@{\hbox}{\bb1@ams@tagbox\hbox}%
6126         \bb1@sreplace\maketag@@@{\hbox}{\bb1@ams@tagbox#1}%
6127         \fi}%
6128 \ifnum\bb1@eqnpos=\tw@%else
6129 \def\bb1@ams@equation{%
6130     \ifnum\bb1@thetextdir>\z@
6131     \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6132     \chardef\bb1@thetextdir\z@
6133     \bb1@add\normalfont{\bb1@eqnodir}%
6134     \ifcase\bb1@eqnpos
6135     \def\veqno##1##2{\bb1@eqno@flip{##1##2}}%
6136     \or
6137     \def\veqno##1##2{\bb1@leqno@flip{##1##2}}%
6138     \fi
6139     \fi}%
6140 \AddToHook{env/equation/begin}{\bb1@ams@equation}%
6141 \AddToHook{env/equation*/begin}{\bb1@ams@equation}%
6142 \fi
6143 \AddToHook{env/cases/begin}{\bb1@ams@preset\bb1@ams@lap}%
6144 \AddToHook{env/multline/begin}{\bb1@ams@preset\hbox}%
6145 \AddToHook{env/gather/begin}{\bb1@ams@preset\bb1@ams@lap}%
6146 \AddToHook{env/gather*/begin}{\bb1@ams@preset\bb1@ams@lap}%
6147 \AddToHook{env/align/begin}{\bb1@ams@preset\bb1@ams@lap}%
6148 \AddToHook{env/align*/begin}{\bb1@ams@preset\bb1@ams@lap}%
6149 \AddToHook{env/eqnalign/begin}{\bb1@ams@preset\hbox}%
6150 % Hackish, for proper alignment. Don't ask me why it works!:
6151 \bb1@exp{% Avoid a 'visible' conditional
6152     \\\AddToHook{env/align*/end}{\<iftag>\<else>\\tag*{\<fi>}}%
6153 \AddToHook{env/flalign/begin}{\bb1@ams@preset\hbox}%
6154 \AddToHook{env/split/before}{%
6155     \ifnum\bb1@thetextdir>\z@
6156     \bb1@ifsamestring\@currentenv{equation}%
6157     {\ifx\bb1@ams@lap\hbox % leqno
6158     \def\bb1@ams@flip#1{%
6159         \hbox to 0.01pt{\hbox to\displaywidth{\hss\hss}}%
6160     \else
6161     \def\bb1@ams@flip#1{%
6162         \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss\hss}}%
6163     \fi}%
6164     }%
6165     \fi}%
6166 \fi
6167 \fi}
6168 \fi
6169 \ifx\bb1@opt@layout\@nnil\endinput\fi % if no layout
6170 \ifnum\bb1@bidimode>\z@
6171 \def\bb1@nextfake#1{% non-local changes, use always inside a group!
6172     \bb1@exp{%
6173         \def\\bb1@insidemath{0}%
6174         \mathdir\the\bodydir
6175         #1% Once entered in math, set boxes to restore values
6176         \<ifmmode>%
6177         \everyvbox{%
6178             \the\everyvbox
6179             \bodydir\the\bodydir
6180             \mathdir\the\mathdir
6181             \everyhbox{\the\everyhbox}%
6182             \everyvbox{\the\everyvbox}}%
6183         \everyhbox{%
6184             \the\everyhbox
6185             \bodydir\the\bodydir

```



```

6186         \mathdir\the\mathdir
6187         \everyhbox{\the\everyhbox}%
6188         \everyvbox{\the\everyvbox}}%
6189     \<fi>}}%
6190 \def\@hangfrom#1{%
6191     \setbox\@tempboxa\hbox{\{#1\}}%
6192     \hangindent\wd\@tempboxa
6193     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6194         \shapemode\@ne
6195     \fi
6196     \noindent\box\@tempboxa}
6197 \fi
6198 \IfBabelLayout{tabular}
6199 {\let\bbl@OL@@tabular\@tabular
6200  \bbl@replace\@tabular{\$}\bbl@nextfake$}%
6201  \let\bbl@NL@@tabular\@tabular
6202  \AtBeginDocument{%
6203      \ifx\bbl@NL@@tabular\@tabular\else
6204          \bbl@replace\@tabular{\$}\bbl@nextfake$}%
6205      \let\bbl@NL@@tabular\@tabular
6206      \fi}}
6207 {}
6208 \IfBabelLayout{lists}
6209 {\let\bbl@OL@list\list
6210  \bbl@sreplace\list{\parshape}\bbl@listparshape}%
6211  \let\bbl@NL@list\list
6212  \def\bbl@listparshape#1#2#3{%
6213      \parshape #1 #2 #3 %
6214      \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6215          \shapemode\tw@
6216      \fi}}
6217 {}
6218 \IfBabelLayout{graphics}
6219 {\let\bbl@pictresetdir\relax
6220  \def\bbl@pictsetdir#1{%
6221      \ifcase\bbl@thetextdir
6222          \let\bbl@pictresetdir\relax
6223      \else
6224          \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6225              \or\textdir TLT
6226              \else\bodydir TLT \textdir TLT
6227          \fi
6228          % \(\text|par)dir required in pgf:
6229          \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6230      \fi}%
6231  \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6232  \directlua{
6233      Babel.get_picture_dir = true
6234      Babel.picture_has_bidi = 0
6235      %
6236      function Babel.picture_dir (head)
6237          if not Babel.get_picture_dir then return head end
6238          if Babel.hlist_has_bidi(head) then
6239              Babel.picture_has_bidi = 1
6240          end
6241          return head
6242      end
6243      luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6244          "Babel.picture_dir")
6245  }%
6246  \AtBeginDocument{%
6247      \long\def\put(#1,#2)#3{%
6248          \@killglue

```

```

6249 % Try:
6250 \ifx\bbbl@pictresetdir\relax
6251 \def\bbbl@tempc{0}%
6252 \else
6253 \directlua{
6254 Babel.get_picture_dir = true
6255 Babel.picture_has_bidi = 0
6256 }%
6257 \setbox\z@\hb@xt@\z@{%
6258 \@defaultunitsset\@tempdimc{#1}\unitlength
6259 \kern\@tempdimc
6260 #3\hss}% TODO: #3 executed twice (below). That's bad.
6261 \edef\bbbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6262 \fi
6263 % Do:
6264 \@defaultunitsset\@tempdimc{#2}\unitlength
6265 \raise\@tempdimc\hb@xt@\z@{%
6266 \@defaultunitsset\@tempdimc{#1}\unitlength
6267 \kern\@tempdimc
6268 {\ifnum\bbbl@tempc>\z@\bbbl@pictresetdir\fi#3}\hss}%
6269 \ignorespaces}%
6270 \MakeRobust\put}%
6271 \AtBeginDocument
6272 {\AddToHook{cmd/diagbox@pict/before}{\let\bbbl@pictsetdir\@gobble}%
6273 \ifx\pgfpicture\@undefined\else % TODO. Allow deactivate?
6274 \AddToHook{env/pgfpicture/begin}{\bbbl@pictsetdir\@ne}%
6275 \bbbl@add\pgfinterruptpicture{\bbbl@pictresetdir}%
6276 \bbbl@add\pgfsys@beginpicture{\bbbl@pictsetdir\z@}%
6277 \fi
6278 \ifx\tikzpicture\@undefined\else
6279 \AddToHook{env/tikzpicture/begin}{\bbbl@pictsetdir\z@}%
6280 \bbbl@add\tikz@atbegin@node{\bbbl@pictresetdir}%
6281 \bbbl@sreplace\tikz{\begingroup}{\begingroup\bbbl@pictsetdir\tw@}%
6282 \fi
6283 \ifx\tcolorbox\@undefined\else
6284 \def\tcb@drawing@env@begin{%
6285 \csname tcb@before@tcb@split@state\endcsname
6286 \bbbl@pictsetdir\tw@
6287 \begin{\kv tcb@graphenv}%
6288 \tcb@bbdraw%
6289 \tcb@apply@graph@patches
6290 }%
6291 \def\tcb@drawing@env@end{%
6292 \end{\kv tcb@graphenv}%
6293 \bbbl@pictresetdir
6294 \csname tcb@after@tcb@split@state\endcsname
6295 }%
6296 \fi
6297 }}
6298 {}

```

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.

```

6299 \IfBabelLayout{counters}%
6300 {\let\bbbl@OL@textsuperscript\@textsuperscript
6301 \bbbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6302 \let\bbbl@latin@arabic=\@arabic
6303 \let\bbbl@OL@arabic\@arabic
6304 \def\@arabic#1{\babelsublr{\bbbl@latin@arabic#1}}%
6305 \ifpackagewith{babel}{bidi=default}%
6306 {\let\bbbl@asciroman=\@roman
6307 \let\bbbl@OL@roman\@roman

```

```

6308 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
6309 \let\bbl@asciiRoman=\@Roman
6310 \let\bbl@OL@roman=\@Roman
6311 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6312 \let\bbl@OL@labelenumii\labelenumii
6313 \def\labelenumii{\theenumii}%
6314 \let\bbl@OL@p@enumiii\p@enumiii
6315 \def\p@enumiii{\p@enumii}\theenumii{}\}\}\}
6316 <<Footnote changes>>
6317 \IfBabelLayout{footnotes}%
6318 {\let\bbl@OL@footnote\footnote
6319 \BabelFootnote\footnote\language\{}}%
6320 \BabelFootnote\localfootnote\language\{}}%
6321 \BabelFootnote\mainfootnote\{}}\}
6322 {}

```

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

```

6323 \IfBabelLayout{extras}%
6324 {\let\bbl@OL@underline\underline
6325 \bbl@sreplace\underline{\$ \@underline}{\bbl@nextfake\$ \@underline}%
6326 \let\bbl@OL@LaTeX2e\LaTeX2e
6327 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6328 \if b\expandafter\car\@series\@nil\boldmath\fi
6329 \babelsublr{%
6330 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}%
6331 {}
6332 </luatex>

```

## 12.11 Lua: transforms

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

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

```

6333 <*transforms>
6334 Babel.linebreaking.replacements = {}
6335 Babel.linebreaking.replacements[0] = {} -- pre
6336 Babel.linebreaking.replacements[1] = {} -- post
6337 Babel.linebreaking.replacements[2] = {} -- post-line WIP
6338
6339 -- Discretionaries contain strings as nodes
6340 function Babel.str_to_nodes(fn, matches, base)
6341 local n, head, last
6342 if fn == nil then return nil end
6343 for s in string.utfvalues(fn(matches)) do
6344 if base.id == 7 then
6345 base = base.replace
6346 end
6347 n = node.copy(base)
6348 n.char = s
6349 if not head then
6350 head = n
6351 else
6352 last.next = n
6353 end

```

```

6354     last = n
6355 end
6356 return head
6357 end
6358
6359 Babel.fetch_subtext = {}
6360
6361 Babel.ignore_pre_char = function(node)
6362     return (node.lang == Babel.nohyphenation)
6363 end
6364
6365 -- Merging both functions doesn't seem feasible, because there are too
6366 -- many differences.
6367 Babel.fetch_subtext[0] = function(head)
6368     local word_string = ''
6369     local word_nodes = {}
6370     local lang
6371     local item = head
6372     local inmath = false
6373
6374     while item do
6375
6376         if item.id == 11 then
6377             inmath = (item.subtype == 0)
6378         end
6379
6380         if inmath then
6381             -- pass
6382         elseif item.id == 29 then
6383             local locale = node.get_attribute(item, Babel.attr_locale)
6384
6385             if lang == locale or lang == nil then
6386                 lang = lang or locale
6387                 if Babel.ignore_pre_char(item) then
6388                     word_string = word_string .. Babel.us_char
6389                 else
6390                     word_string = word_string .. unicode.utf8.char(item.char)
6391                 end
6392                 word_nodes[#word_nodes+1] = item
6393             else
6394                 break
6395             end
6396         elseif item.id == 12 and item.subtype == 13 then
6397             word_string = word_string .. ' '
6398             word_nodes[#word_nodes+1] = item
6399         end
6400
6401         -- Ignore leading unrecognized nodes, too.
6402         elseif word_string ~= '' then
6403             word_string = word_string .. Babel.us_char
6404             word_nodes[#word_nodes+1] = item -- Will be ignored
6405         end
6406
6407         item = item.next
6408     end
6409
6410     -- Here and above we remove some trailing chars but not the
6411     -- corresponding nodes. But they aren't accessed.
6412     if word_string:sub(-1) == ' ' then
6413         word_string = word_string:sub(1,-2)
6414     end
6415     word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')

```

```

6417 return word_string, word_nodes, item, lang
6418 end
6419
6420 Babel.fetch_subtext[1] = function(head)
6421   local word_string = ''
6422   local word_nodes = {}
6423   local lang
6424   local item = head
6425   local inmath = false
6426
6427   while item do
6428
6429     if item.id == 11 then
6430       inmath = (item.subtype == 0)
6431     end
6432
6433     if inmath then
6434       -- pass
6435
6436     elseif item.id == 29 then
6437       if item.lang == lang or lang == nil then
6438         if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6439           lang = lang or item.lang
6440           word_string = word_string .. unicode.utf8.char(item.char)
6441           word_nodes[#word_nodes+1] = item
6442         end
6443       else
6444         break
6445       end
6446
6447     elseif item.id == 7 and item.subtype == 2 then
6448       word_string = word_string .. '='
6449       word_nodes[#word_nodes+1] = item
6450
6451     elseif item.id == 7 and item.subtype == 3 then
6452       word_string = word_string .. '|'
6453       word_nodes[#word_nodes+1] = item
6454
6455     -- (1) Go to next word if nothing was found, and (2) implicitly
6456     -- remove leading USs.
6457     elseif word_string == '' then
6458       -- pass
6459
6460     -- This is the responsible for splitting by words.
6461     elseif (item.id == 12 and item.subtype == 13) then
6462       break
6463
6464     else
6465       word_string = word_string .. Babel.us_char
6466       word_nodes[#word_nodes+1] = item -- Will be ignored
6467     end
6468
6469     item = item.next
6470   end
6471
6472   word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6473   return word_string, word_nodes, item, lang
6474 end
6475
6476 function Babel.pre_hyphenate_replace(head)
6477   Babel.hyphenate_replace(head, 0)
6478 end
6479

```

```

6480 function Babel.post_hyphenate_replace(head)
6481   Babel.hyphenate_replace(head, 1)
6482 end
6483
6484 Babel.us_char = string.char(31)
6485
6486 function Babel.hyphenate_replace(head, mode)
6487   local u = unicode.utf8
6488   local lbkr = Babel.linebreaking.replacements[mode]
6489   if mode == 2 then mode = 0 end -- WIP
6490
6491   local word_head = head
6492
6493   while true do -- for each subtext block
6494
6495     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6496
6497     if Babel.debug then
6498       print()
6499       print((mode == 0) and '@@@<' or '@@@>', w)
6500     end
6501
6502     if nw == nil and w == '' then break end
6503
6504     if not lang then goto next end
6505     if not lbkr[lang] then goto next end
6506
6507     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6508     -- loops are nested.
6509     for k=1, #lbkr[lang] do
6510       local p = lbkr[lang][k].pattern
6511       local r = lbkr[lang][k].replace
6512       local attr = lbkr[lang][k].attr or -1
6513
6514       if Babel.debug then
6515         print('*****', p, mode)
6516       end
6517
6518       -- This variable is set in some cases below to the first *byte*
6519       -- after the match, either as found by u.match (faster) or the
6520       -- computed position based on sc if w has changed.
6521       local last_match = 0
6522       local step = 0
6523
6524       -- For every match.
6525       while true do
6526         if Babel.debug then
6527           print('====')
6528         end
6529         local new -- used when inserting and removing nodes
6530
6531         local matches = { u.match(w, p, last_match) }
6532
6533         if #matches < 2 then break end
6534
6535         -- Get and remove empty captures (with ())'s, which return a
6536         -- number with the position), and keep actual captures
6537         -- (from (...)), if any, in matches.
6538         local first = table.remove(matches, 1)
6539         local last = table.remove(matches, #matches)
6540         -- Non re-fetched substrings may contain \31, which separates
6541         -- subsubstrings.
6542         if string.find(w:sub(first, last-1), Babel.us_char) then break end

```

```

6543
6544     local save_last = last -- with A()BC()D, points to D
6545
6546     -- Fix offsets, from bytes to unicode. Explained above.
6547     first = u.len(w:sub(1, first-1)) + 1
6548     last  = u.len(w:sub(1, last-1)) -- now last points to C
6549
6550     -- This loop stores in a small table the nodes
6551     -- corresponding to the pattern. Used by 'data' to provide a
6552     -- predictable behavior with 'insert' (w_nodes is modified on
6553     -- the fly), and also access to 'remove'd nodes.
6554     local sc = first-1 -- Used below, too
6555     local data_nodes = {}
6556
6557     local enabled = true
6558     for q = 1, last-first+1 do
6559         data_nodes[q] = w_nodes[sc+q]
6560         if enabled
6561             and attr > -1
6562             and not node.has_attribute(data_nodes[q], attr)
6563         then
6564             enabled = false
6565         end
6566     end
6567
6568     -- This loop traverses the matched substring and takes the
6569     -- corresponding action stored in the replacement list.
6570     -- sc = the position in substr nodes / string
6571     -- rc = the replacement table index
6572     local rc = 0
6573
6574     while rc < last-first+1 do -- for each replacement
6575         if Babel.debug then
6576             print('....', rc + 1)
6577         end
6578         sc = sc + 1
6579         rc = rc + 1
6580
6581         if Babel.debug then
6582             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6583             local ss = ''
6584             for itt in node.traverse(head) do
6585                 if itt.id == 29 then
6586                     ss = ss .. unicode.utf8.char(itt.char)
6587                 else
6588                     ss = ss .. '{' .. itt.id .. '}'
6589                 end
6590             end
6591             print('*****', ss)
6592         end
6593
6594         local crep = r[rc]
6595         local item = w_nodes[sc]
6596         local item_base = item
6597         local placeholder = Babel.us_char
6598         local d
6599
6600         if crep and crep.data then
6601             item_base = data_nodes[crep.data]
6602         end
6603
6604         if crep then

```

```

6606         step = crep.step or 0
6607     end
6608
6609     if (not enabled) or (crep and next(crep) == nil) then -- = {}
6610         last_match = save_last    -- Optimization
6611         goto next
6612
6613     elseif crep == nil or crep.remove then
6614         node.remove(head, item)
6615         table.remove(w_nodes, sc)
6616         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6617         sc = sc - 1 -- Nothing has been inserted.
6618         last_match = utf8.offset(w, sc+1+step)
6619         goto next
6620
6621     elseif crep and crep.kashida then -- Experimental
6622         node.set_attribute(item,
6623             Babel.attr_kashida,
6624             crep.kashida)
6625         last_match = utf8.offset(w, sc+1+step)
6626         goto next
6627
6628     elseif crep and crep.string then
6629         local str = crep.string(matches)
6630         if str == '' then -- Gather with nil
6631             node.remove(head, item)
6632             table.remove(w_nodes, sc)
6633             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6634             sc = sc - 1 -- Nothing has been inserted.
6635         else
6636             local loop_first = true
6637             for s in string.utfvalues(str) do
6638                 d = node.copy(item_base)
6639                 d.char = s
6640                 if loop_first then
6641                     loop_first = false
6642                     head, new = node.insert_before(head, item, d)
6643                     if sc == 1 then
6644                         word_head = head
6645                     end
6646                     w_nodes[sc] = d
6647                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6648                 else
6649                     sc = sc + 1
6650                     head, new = node.insert_before(head, item, d)
6651                     table.insert(w_nodes, sc, new)
6652                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6653                 end
6654                 if Babel.debug then
6655                     print('.....', 'str')
6656                     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6657                 end
6658             end -- for
6659             node.remove(head, item)
6660         end -- if ''
6661         last_match = utf8.offset(w, sc+1+step)
6662         goto next
6663
6664     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6665         d = node.new(7, 0) -- (disc, discretionary)
6666         d.pre      = Babel.str_to_nodes(crep.pre, matches, item_base)
6667         d.post     = Babel.str_to_nodes(crep.post, matches, item_base)
6668         d.replace = Babel.str_to_nodes(crep.no, matches, item_base)

```



```

6669         d.attr = item_base.attr
6670         if crep.pre == nil then -- TeXbook p96
6671             d.penalty = crep.penalty or tex.hyphenpenalty
6672         else
6673             d.penalty = crep.penalty or tex.exhyphenpenalty
6674         end
6675         placeholder = '|'
6676         head, new = node.insert_before(head, item, d)
6677
6678     elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6679         -- ERROR
6680
6681     elseif crep and crep.penalty then
6682         d = node.new(14, 0) -- (penalty, userpenalty)
6683         d.attr = item_base.attr
6684         d.penalty = crep.penalty
6685         head, new = node.insert_before(head, item, d)
6686
6687     elseif crep and crep.space then
6688         -- 655360 = 10 pt = 10 * 65536 sp
6689         d = node.new(12, 13) -- (glue, spaceskip)
6690         local quad = font.getfont(item_base.font).size or 655360
6691         node.setglue(d, crep.space[1] * quad,
6692                     crep.space[2] * quad,
6693                     crep.space[3] * quad)
6694         if mode == 0 then
6695             placeholder = ' '
6696         end
6697         head, new = node.insert_before(head, item, d)
6698
6699     elseif crep and crep.spacefactor then
6700         d = node.new(12, 13) -- (glue, spaceskip)
6701         local base_font = font.getfont(item_base.font)
6702         node.setglue(d,
6703                     crep.spacefactor[1] * base_font.parameters['space'],
6704                     crep.spacefactor[2] * base_font.parameters['space_stretch'],
6705                     crep.spacefactor[3] * base_font.parameters['space_shrink'])
6706         if mode == 0 then
6707             placeholder = ' '
6708         end
6709         head, new = node.insert_before(head, item, d)
6710
6711     elseif mode == 0 and crep and crep.space then
6712         -- ERROR
6713
6714     end -- ie replacement cases
6715
6716     -- Shared by disc, space and penalty.
6717     if sc == 1 then
6718         word_head = head
6719     end
6720     if crep.insert then
6721         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6722         table.insert(w_nodes, sc, new)
6723         last = last + 1
6724     else
6725         w_nodes[sc] = d
6726         node.remove(head, item)
6727         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6728     end
6729
6730     last_match = utf8.offset(w, sc+1+step)
6731

```

```

6732         ::next::
6733
6734     end -- for each replacement
6735
6736     if Babel.debug then
6737         print('.....', '/')
6738         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6739     end
6740
6741     end -- for match
6742
6743     end -- for patterns
6744
6745     ::next::
6746     word_head = nw
6747 end -- for substring
6748 return head
6749 end
6750
6751 -- This table stores capture maps, numbered consecutively
6752 Babel.capture_maps = {}
6753
6754 -- The following functions belong to the next macro
6755 function Babel.capture_func(key, cap)
6756     local ret = "[" .. cap:gsub('{{([0-9])}}', "]]..m[%1]..[") .. "]"
6757     local cnt
6758     local u = unicode.utf8
6759     ret, cnt = ret:gsub('{{([0-9])|([^\]]+)|(.-}}', Babel.capture_func_map)
6760     if cnt == 0 then
6761         ret = u.gsub(ret, '{{(%x%x%x%x+}}',
6762             function (n)
6763                 return u.char(tonumber(n, 16))
6764             end)
6765     end
6766     ret = ret:gsub("%[%[%]]%.", '')
6767     ret = ret:gsub("%.%[%[%]]%", '')
6768     return key .. [[=function(m) return ]] .. ret .. [[ end]]
6769 end
6770
6771 function Babel.capt_map(from, mapno)
6772     return Babel.capture_maps[mapno][from] or from
6773 end
6774
6775 -- Handle the {n|abc|ABC} syntax in captures
6776 function Babel.capture_func_map(capno, from, to)
6777     local u = unicode.utf8
6778     from = u.gsub(from, '{{(%x%x%x%x+}}',
6779         function (n)
6780             return u.char(tonumber(n, 16))
6781         end)
6782     to = u.gsub(to, '{{(%x%x%x%x+}}',
6783         function (n)
6784             return u.char(tonumber(n, 16))
6785         end)
6786     local froms = {}
6787     for s in string.utfcharacters(from) do
6788         table.insert(froms, s)
6789     end
6790     local cnt = 1
6791     table.insert(Babel.capture_maps, {})
6792     local mlen = table.getn(Babel.capture_maps)
6793     for s in string.utfcharacters(to) do
6794         Babel.capture_maps[mlen][froms[cnt]] = s

```

```

6795     cnt = cnt + 1
6796 end
6797 return "]]..Babel.capt_map(m[" .. capno .. "]," ..
6798         (mlen) .. ").." .. "["
6799 end
6800
6801 -- Create/Extend reversed sorted list of kashida weights:
6802 function Babel.capture_kashida(key, wt)
6803     wt = tonumber(wt)
6804     if Babel.kashida_wts then
6805         for p, q in ipairs(Babel.kashida_wts) do
6806             if wt == q then
6807                 break
6808             elseif wt > q then
6809                 table.insert(Babel.kashida_wts, p, wt)
6810                 break
6811             elseif table.getn(Babel.kashida_wts) == p then
6812                 table.insert(Babel.kashida_wts, wt)
6813             end
6814         end
6815     else
6816         Babel.kashida_wts = { wt }
6817     end
6818     return 'kashida = ' .. wt
6819 end
6820 </transforms>

```

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

```

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

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

Arrrrgh!! 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.

```

6821 <*basic-r>
6822 Babel = Babel or {}
6823
6824 Babel.bidi_enabled = true
6825
6826 require('babel-data-bidi.lua')
6827
6828 local characters = Babel.characters
6829 local ranges = Babel.ranges
6830
6831 local DIR = node.id("dir")
6832
6833 local function dir_mark(head, from, to, outer)
6834   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6835   local d = node.new(DIR)
6836   d.dir = '+' .. dir
6837   node.insert_before(head, from, d)
6838   d = node.new(DIR)
6839   d.dir = '-' .. dir
6840   node.insert_after(head, to, d)
6841 end
6842
6843 function Babel.bidi(head, ispar)
6844   local first_n, last_n          -- first and last char with nums
6845   local last_es                  -- an auxiliary 'last' used with nums
6846   local first_d, last_d          -- first and last char in L/R block
6847   local dir, dir_real

```

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

```

6848   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6849   local strong_lr = (strong == 'l') and 'l' or 'r'
6850   local outer = strong
6851
6852   local new_dir = false
6853   local first_dir = false
6854   local inmath = false
6855
6856   local last_lr
6857
6858   local type_n = ''
6859
6860   for item in node.traverse(head) do
6861
6862     -- three cases: glyph, dir, otherwise
6863     if item.id == node.id'glyph'
6864       or (item.id == 7 and item.subtype == 2) then
6865
6866       local itemchar
6867       if item.id == 7 and item.subtype == 2 then
6868         itemchar = item.replace.char
6869       else
6870         itemchar = item.char
6871       end
6872       local chardata = characters[itemchar]
6873       dir = chardata and chardata.d or nil
6874       if not dir then
6875         for nn, et in ipairs(ranges) do
6876           if itemchar < et[1] then
6877             break

```

```

6878         elseif itemchar <= et[2] then
6879             dir = et[3]
6880             break
6881         end
6882     end
6883 end
6884 dir = dir or 'l'
6885 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).

```

6886 if new_dir then
6887     attr_dir = 0
6888     for at in node.traverse(item.attr) do
6889         if at.number == Babel.attr_dir then
6890             attr_dir = at.value % 3
6891         end
6892     end
6893     if attr_dir == 1 then
6894         strong = 'r'
6895     elseif attr_dir == 2 then
6896         strong = 'al'
6897     else
6898         strong = 'l'
6899     end
6900     strong_lr = (strong == 'l') and 'l' or 'r'
6901     outer = strong_lr
6902     new_dir = false
6903 end
6904
6905 if dir == 'nsm' then dir = strong end -- W1

```

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

```

6906 dir_real = dir -- We need dir_real to set strong below
6907 if dir == 'al' then dir = 'r' end -- W3

```

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

```

6908 if strong == 'al' then
6909     if dir == 'en' then dir = 'an' end -- W2
6910     if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6911     strong_lr = 'r' -- W3
6912 end

```

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

```

6913 elseif item.id == node.id'dir' and not inmath then
6914     new_dir = true
6915     dir = nil
6916 elseif item.id == node.id'math' then
6917     inmath = (item.subtype == 0)
6918 else
6919     dir = nil -- Not a char
6920 end

```

Numbers in R mode. A sequence of <en>, <et>, <an>, <es> and <cs> is typeset (with some rules) in L mode. We store the starting and ending points, and only when anything different is found (including nil, ie, a non-char), the textdir is set. This means you cannot insert, say, a whatsit, but this is what I would expect (with luacolor you may colorize some digits). Anyway, this behavior could be changed with a switch in the future. Note in the first branch only <an> is relevant if <al>.

```

6921 if dir == 'en' or dir == 'an' or dir == 'et' then
6922     if dir ~= 'et' then

```

```

6923     type_n = dir
6924 end
6925 first_n = first_n or item
6926 last_n = last_es or item
6927 last_es = nil
6928 elseif dir == 'es' and last_n then -- W3+W6
6929     last_es = item
6930 elseif dir == 'cs' then          -- it's right - do nothing
6931 elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6932     if strong_lr == 'r' and type_n ~= '' then
6933         dir_mark(head, first_n, last_n, 'r')
6934     elseif strong_lr == 'l' and first_d and type_n == 'an' then
6935         dir_mark(head, first_n, last_n, 'r')
6936         dir_mark(head, first_d, last_d, outer)
6937         first_d, last_d = nil, nil
6938     elseif strong_lr == 'l' and type_n ~= '' then
6939         last_d = last_n
6940     end
6941     type_n = ''
6942     first_n, last_n = nil, nil
6943 end

```

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

```

6944 if dir == 'l' or dir == 'r' then
6945     if dir ~= outer then
6946         first_d = first_d or item
6947         last_d = item
6948     elseif first_d and dir ~= strong_lr then
6949         dir_mark(head, first_d, last_d, outer)
6950         first_d, last_d = nil, nil
6951     end
6952 end

```

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

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

```

6953 if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6954     item.char = characters[item.char] and
6955         characters[item.char].m or item.char
6956 elseif (dir or new_dir) and last_lr ~= item then
6957     local mir = outer .. strong_lr .. (dir or outer)
6958     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6959         for ch in node.traverse(node.next(last_lr)) do
6960             if ch == item then break end
6961             if ch.id == node.id'glyph' and characters[ch.char] then
6962                 ch.char = characters[ch.char].m or ch.char
6963             end
6964         end
6965     end
6966 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).

```

6967 if dir == 'l' or dir == 'r' then
6968     last_lr = item
6969     strong = dir_real          -- Don't search back - best save now
6970     strong_lr = (strong == 'l') and 'l' or 'r'
6971 elseif new_dir then
6972     last_lr = nil

```

```

6973     end
6974 end

```

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

```

6975 if last_lr and outer == 'r' then
6976     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6977         if characters[ch.char] then
6978             ch.char = characters[ch.char].m or ch.char
6979         end
6980     end
6981 end
6982 if first_n then
6983     dir_mark(head, first_n, last_n, outer)
6984 end
6985 if first_d then
6986     dir_mark(head, first_d, last_d, outer)
6987 end

```

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

```

6988 return node.prev(head) or head
6989 end
6990 </basic-r>

```

And here the Lua code for bidi=basic:

```

6991 <*basic>
6992 Babel = Babel or {}
6993
6994 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6995
6996 Babel.fontmap = Babel.fontmap or {}
6997 Babel.fontmap[0] = {}      -- l
6998 Babel.fontmap[1] = {}      -- r
6999 Babel.fontmap[2] = {}      -- al/an
7000
7001 Babel.bidi_enabled = true
7002 Babel.mirroring_enabled = true
7003
7004 require('babel-data-bidi.lua')
7005
7006 local characters = Babel.characters
7007 local ranges = Babel.ranges
7008
7009 local DIR = node.id('dir')
7010 local GLYPH = node.id('glyph')
7011
7012 local function insert_implicit(head, state, outer)
7013     local new_state = state
7014     if state.sim and state.eim and state.sim ~= state.eim then
7015         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
7016         local d = node.new(DIR)
7017         d.dir = '+' .. dir
7018         node.insert_before(head, state.sim, d)
7019         local d = node.new(DIR)
7020         d.dir = '-' .. dir
7021         node.insert_after(head, state.eim, d)
7022     end
7023     new_state.sim, new_state.eim = nil, nil
7024     return head, new_state
7025 end
7026
7027 local function insert_numeric(head, state)
7028     local new
7029     local new_state = state

```

```

7030 if state.san and state.ean and state.san ~= state.ean then
7031     local d = node.new(DIR)
7032     d.dir = '+TLT'
7033     _, new = node.insert_before(head, state.san, d)
7034     if state.san == state.sim then state.sim = new end
7035     local d = node.new(DIR)
7036     d.dir = '-TLT'
7037     _, new = node.insert_after(head, state.ean, d)
7038     if state.ean == state.eim then state.eim = new end
7039 end
7040 new_state.san, new_state.ean = nil, nil
7041 return head, new_state
7042 end
7043
7044 -- TODO - \hbox with an explicit dir can lead to wrong results
7045 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7046 -- was s made to improve the situation, but the problem is the 3-dir
7047 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7048 -- well.
7049
7050 function Babel.bidi(head, ispar, hdir)
7051     local d -- d is used mainly for computations in a loop
7052     local prev_d = ''
7053     local new_d = false
7054
7055     local nodes = {}
7056     local outer_first = nil
7057     local inmath = false
7058
7059     local glue_d = nil
7060     local glue_i = nil
7061
7062     local has_en = false
7063     local first_et = nil
7064
7065     local ATDIR = Babel.attr_dir
7066
7067     local save_outer
7068     local temp = node.get_attribute(head, ATDIR)
7069     if temp then
7070         temp = temp % 3
7071         save_outer = (temp == 0 and 'l') or
7072             (temp == 1 and 'r') or
7073             (temp == 2 and 'al')
7074     elseif ispar then -- Or error? Shouldn't happen
7075         save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7076     else -- Or error? Shouldn't happen
7077         save_outer = ('TRT' == hdir) and 'r' or 'l'
7078     end
7079     -- when the callback is called, we are just _after_ the box,
7080     -- and the textdir is that of the surrounding text
7081     -- if not ispar and hdir ~= tex.textdir then
7082     --     save_outer = ('TRT' == hdir) and 'r' or 'l'
7083     -- end
7084     local outer = save_outer
7085     local last = outer
7086     -- 'al' is only taken into account in the first, current loop
7087     if save_outer == 'al' then save_outer = 'r' end
7088
7089     local fontmap = Babel.fontmap
7090
7091     for item in node.traverse(head) do
7092

```



```

7093 -- In what follows, #node is the last (previous) node, because the
7094 -- current one is not added until we start processing the neutrals.
7095
7096 -- three cases: glyph, dir, otherwise
7097 if item.id == GLYPH
7098     or (item.id == 7 and item.subtype == 2) then
7099
7100     local d_font = nil
7101     local item_r
7102     if item.id == 7 and item.subtype == 2 then
7103         item_r = item.replace -- automatic discs have just 1 glyph
7104     else
7105         item_r = item
7106     end
7107     local chardata = characters[item_r.char]
7108     d = chardata and chardata.d or nil
7109     if not d or d == 'nsm' then
7110         for nn, et in ipairs(ranges) do
7111             if item_r.char < et[1] then
7112                 break
7113             elseif item_r.char <= et[2] then
7114                 if not d then d = et[3]
7115                 elseif d == 'nsm' then d_font = et[3]
7116             end
7117             break
7118         end
7119     end
7120     end
7121     d = d or 'l'
7122
7123     -- A short 'pause' in bidi for mapfont
7124     d_font = d_font or d
7125     d_font = (d_font == 'l' and 0) or
7126             (d_font == 'nsm' and 0) or
7127             (d_font == 'r' and 1) or
7128             (d_font == 'al' and 2) or
7129             (d_font == 'an' and 2) or nil
7130     if d_font and fontmap and fontmap[d_font][item_r.font] then
7131         item_r.font = fontmap[d_font][item_r.font]
7132     end
7133
7134     if new_d then
7135         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7136         if inmath then
7137             attr_d = 0
7138         else
7139             attr_d = node.get_attribute(item, ATDIR)
7140             attr_d = attr_d % 3
7141         end
7142         if attr_d == 1 then
7143             outer_first = 'r'
7144             last = 'r'
7145         elseif attr_d == 2 then
7146             outer_first = 'r'
7147             last = 'al'
7148         else
7149             outer_first = 'l'
7150             last = 'l'
7151         end
7152         outer = last
7153         has_en = false
7154         first_et = nil
7155         new_d = false

```

```

7156     end
7157
7158     if glue_d then
7159         if (d == 'l' and 'l' or 'r') ~= glue_d then
7160             table.insert(nodes, {glue_i, 'on', nil})
7161         end
7162         glue_d = nil
7163         glue_i = nil
7164     end
7165
7166     elseif item.id == DIR then
7167         d = nil
7168         if head ~= item then new_d = true end
7169
7170     elseif item.id == node.id'glue' and item.subtype == 13 then
7171         glue_d = d
7172         glue_i = item
7173         d = nil
7174
7175     elseif item.id == node.id'math' then
7176         inmath = (item.subtype == 0)
7177
7178     else
7179         d = nil
7180     end
7181
7182     -- AL <= EN/ET/ES      -- W2 + W3 + W6
7183     if last == 'al' and d == 'en' then
7184         d = 'an'          -- W3
7185     elseif last == 'al' and (d == 'et' or d == 'es') then
7186         d = 'on'          -- W6
7187     end
7188
7189     -- EN + CS/ES + EN      -- W4
7190     if d == 'en' and #nodes >= 2 then
7191         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7192             and nodes[#nodes-1][2] == 'en' then
7193             nodes[#nodes][2] = 'en'
7194         end
7195     end
7196
7197     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
7198     if d == 'an' and #nodes >= 2 then
7199         if (nodes[#nodes][2] == 'cs')
7200             and nodes[#nodes-1][2] == 'an' then
7201             nodes[#nodes][2] = 'an'
7202         end
7203     end
7204
7205     -- ET/EN                -- W5 + W7->l / W6->on
7206     if d == 'et' then
7207         first_et = first_et or (#nodes + 1)
7208     elseif d == 'en' then
7209         has_en = true
7210         first_et = first_et or (#nodes + 1)
7211     elseif first_et then    -- d may be nil here !
7212         if has_en then
7213             if last == 'l' then
7214                 temp = 'l'    -- W7
7215             else
7216                 temp = 'en'   -- W5
7217             end
7218         else

```

```

7219         temp = 'on'      -- W6
7220     end
7221     for e = first_et, #nodes do
7222         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7223     end
7224     first_et = nil
7225     has_en = false
7226 end
7227
7228 -- Force mathdir in math if ON (currently works as expected only
7229 -- with 'l')
7230 if inmath and d == 'on' then
7231     d = ('TRT' == tex.mathdir) and 'r' or 'l'
7232 end
7233
7234 if d then
7235     if d == 'al' then
7236         d = 'r'
7237         last = 'al'
7238     elseif d == 'l' or d == 'r' then
7239         last = d
7240     end
7241     prev_d = d
7242     table.insert(nodes, {item, d, outer_first})
7243 end
7244
7245 outer_first = nil
7246
7247 end
7248
7249 -- TODO -- repeated here in case EN/ET is the last node. Find a
7250 -- better way of doing things:
7251 if first_et then      -- dir may be nil here !
7252     if has_en then
7253         if last == 'l' then
7254             temp = 'l'      -- W7
7255         else
7256             temp = 'en'     -- W5
7257         end
7258     else
7259         temp = 'on'        -- W6
7260     end
7261     for e = first_et, #nodes do
7262         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7263     end
7264 end
7265
7266 -- dummy node, to close things
7267 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7268
7269 ----- NEUTRAL -----
7270
7271 outer = save_outer
7272 last = outer
7273
7274 local first_on = nil
7275
7276 for q = 1, #nodes do
7277     local item
7278
7279     local outer_first = nodes[q][3]
7280     outer = outer_first or outer
7281     last = outer_first or last

```

```

7282
7283     local d = nodes[q][2]
7284     if d == 'an' or d == 'en' then d = 'r' end
7285     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7286
7287     if d == 'on' then
7288         first_on = first_on or q
7289     elseif first_on then
7290         if last == d then
7291             temp = d
7292         else
7293             temp = outer
7294         end
7295         for r = first_on, q - 1 do
7296             nodes[r][2] = temp
7297             item = nodes[r][1] -- MIRRORING
7298             if Babel.mirroring_enabled and item.id == GLYPH
7299                 and temp == 'r' and characters[item.char] then
7300                 local font_mode = ''
7301                 if font.fonts[item.font].properties then
7302                     font_mode = font.fonts[item.font].properties.mode
7303                 end
7304                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
7305                     item.char = characters[item.char].m or item.char
7306                 end
7307             end
7308         end
7309         first_on = nil
7310     end
7311
7312     if d == 'r' or d == 'l' then last = d end
7313 end
7314
7315 ----- IMPLICIT, REORDER -----
7316
7317 outer = save_outer
7318 last = outer
7319
7320 local state = {}
7321 state.has_r = false
7322
7323 for q = 1, #nodes do
7324     local item = nodes[q][1]
7325
7326     outer = nodes[q][3] or outer
7327
7328     local d = nodes[q][2]
7329
7330     if d == 'nsm' then d = last end -- W1
7331     if d == 'en' then d = 'an' end
7332     local isdir = (d == 'r' or d == 'l')
7333
7334     if outer == 'l' and d == 'an' then
7335         state.san = state.san or item
7336         state.ean = item
7337     elseif state.san then
7338         head, state = insert_numeric(head, state)
7339     end
7340
7341     if outer == 'l' then
7342         if d == 'an' or d == 'r' then -- im -> implicit
7343             if d == 'r' then state.has_r = true end

```

```

7345     state.sim = state.sim or item
7346     state.eim = item
7347     elseif d == 'l' and state.sim and state.has_r then
7348         head, state = insert_implicit(head, state, outer)
7349     elseif d == 'l' then
7350         state.sim, state.eim, state.has_r = nil, nil, false
7351     end
7352 else
7353     if d == 'an' or d == 'l' then
7354         if nodes[q][3] then -- nil except after an explicit dir
7355             state.sim = item -- so we move sim 'inside' the group
7356         else
7357             state.sim = state.sim or item
7358         end
7359         state.eim = item
7360     elseif d == 'r' and state.sim then
7361         head, state = insert_implicit(head, state, outer)
7362     elseif d == 'r' then
7363         state.sim, state.eim = nil, nil
7364     end
7365 end
7366
7367 if isdir then
7368     last = d -- Don't search back - best save now
7369 elseif d == 'on' and state.san then
7370     state.san = state.san or item
7371     state.ean = item
7372 end
7373
7374 end
7375
7376 return node.prev(head) or head
7377 end
7378 </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'},
[0x0029]={c='cp'},
[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.

```

7379 <*nil>
7380 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language]
7381 \LdfInit{nil}{datenil}

```

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

```

7382 \ifx\l@nil\@undefined
7383   \newlanguage\l@nil
7384   \@namedef{bbl@hyphendata@the\l@nil}{\{}}% Remove warning
7385   \let\bbl@elt\relax
7386   \edef\bbl@languages{% Add it to the list of languages
7387     \bbl@languages\bbl@elt{nil}{the\l@nil}{\{}}
7388 \fi

```

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

```
7389 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```

\captionnil
\datenil
7390 \let\captionsnil\@empty
7391 \let\datenil\@empty

```

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

```

7392 \def\bbl@inidata@nil{%
7393   \bbl@elt{identification}{tag.ini}{und}%
7394   \bbl@elt{identification}{load.level}{0}%
7395   \bbl@elt{identification}{charset}{utf8}%
7396   \bbl@elt{identification}{version}{1.0}%
7397   \bbl@elt{identification}{date}{2022-05-16}%
7398   \bbl@elt{identification}{name.local}{nil}%
7399   \bbl@elt{identification}{name.english}{nil}%
7400   \bbl@elt{identification}{name.babel}{nil}%
7401   \bbl@elt{identification}{tag.bcp47}{und}%
7402   \bbl@elt{identification}{language.tag.bcp47}{und}%
7403   \bbl@elt{identification}{tag.opentype}{dflt}%
7404   \bbl@elt{identification}{script.name}{Latin}%
7405   \bbl@elt{identification}{script.tag.bcp47}{Latn}%
7406   \bbl@elt{identification}{script.tag.opentype}{DFLT}%
7407   \bbl@elt{identification}{level}{1}%
7408   \bbl@elt{identification}{encodings}{}%
7409   \bbl@elt{identification}{derivate}{no}}
7410 \@namedef{bbl@tbc@nil}{und}
7411 \@namedef{bbl@lbc@nil}{und}
7412 \@namedef{bbl@lotf@nil}{dflt}
7413 \@namedef{bbl@elname@nil}{nil}
7414 \@namedef{bbl@lname@nil}{nil}
7415 \@namedef{bbl@esname@nil}{Latin}
7416 \@namedef{bbl@sname@nil}{Latin}
7417 \@namedef{bbl@sbc@nil}{Latn}
7418 \@namedef{bbl@sotf@nil}{Latn}

```

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

```

7419 \ldf@finish{nil}
7420 \</nil>

```

## 15 Calendars

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

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

```

7421 \<Compute Julian day> \equiv
7422 \def\bbl@fpm#1#2{(#1-#2*floor(#1/#2))}
7423 \def\bbl@cs@gregleap#1{%
7424   (\bbl@fpm{#1}{4} == 0) &&
7425   (!((\bbl@fpm{#1}{100} == 0) && (\bbl@fpm{#1}{400} != 0)))}

```

```

7426 \def\bbl@cs@jd#1#2#3{% year, month, day
7427   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
7428     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
7429     floor((#1 - 1) / 400) + floor((((367 * #2) - 362) / 12) +
7430     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3) }}
7431 <</Compute Julian day>>

```

## 15.1 Islamic

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

```

7432 <*ca-islamic>
7433 \ExplSyntaxOn
7434 <<Compute Julian day>>
7435 % == islamic (default)
7436 % Not yet implemented
7437 \def\bbl@ca@islamic#1-#2-#3@@#4#5#6{}

```

The Civil calendar.

```

7438 \def\bbl@cs@isltojd#1#2#3{% year, month, day
7439   ((#3 + ceil(29.5 * (#2 - 1)) +
7440     (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
7441     1948439.5) - 1) }
7442 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+2}}
7443 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
7444 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
7445 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
7446 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
7447 \def\bbl@ca@islamicvl@x#1#2-#3-#4@@#5#6#7{%
7448   \edef\bbl@tempa{%
7449     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1} }%
7450   \edef#5{%
7451     \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
7452   \edef#6{\fp_eval:n{
7453     min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
7454   \edef#7{\fp_eval:n{ \bbl@tempa - \bbl@cs@isltojd{#5}{#6}{1} + 1 } }}

```

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

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

```

7455 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
7456 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
7457 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
7458 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
7459 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
7460 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
7461 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
7462 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
7463 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
7464 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
7465 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
7466 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
7467 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
7468 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
7469 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
7470 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
7471 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
7472 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
7473 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
7474 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
7475 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
7476 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
7477 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%

```

```

7478 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
7479 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
7480 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
7481 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
7482 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
7483 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
7484 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
7485 65401,65431,65460,65490,65520}
7486 \namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
7487 \namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
7488 \namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
7489 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
7490 \ifnum#2>2014 \ifnum#2<2038
7491 \bbl@afterfi\expandafter\gobble
7492 \fi\fi
7493 {\bbl@error{Year-out-of-range}{The-allowed-range-is-2014-2038}}%
7494 \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
7495 \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
7496 \count@\@ne
7497 \bbl@foreach\bbl@cs@umalqura@data{%
7498 \advance\count@\@ne
7499 \ifnum##1>\bbl@tempd\else
7500 \edef\bbl@tempe{\the\count@}%
7501 \edef\bbl@tempb{##1}%
7502 \fi}%
7503 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month-lunar
7504 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1) / 12) }}% annus
7505 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
7506 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
7507 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
7508 \ExplSyntaxOff
7509 \bbl@add\bbl@precalendar{%
7510 \bbl@replace\bbl@ld@calendar{-civil}{}}%
7511 \bbl@replace\bbl@ld@calendar{-umalqura}{}}%
7512 \bbl@replace\bbl@ld@calendar{+}{}}%
7513 \bbl@replace\bbl@ld@calendar{-}{}}%
7514 </ca-islamic>

```

## 16 Hebrew

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

```

7515 <*ca-hebrew>
7516 \newcount\bbl@cntcommon
7517 \def\bbl@remainder#1#2#3{%
7518 #3=#1\relax
7519 \divide #3 by #2\relax
7520 \multiply #3 by -#2\relax
7521 \advance #3 by #1\relax}%
7522 \newif\ifbbl@divisible
7523 \def\bbl@checkifdivisible#1#2{%
7524 {\countdef\tmp=0
7525 \bbl@remainder{#1}{#2}{\tmp}%
7526 \ifnum \tmp=0
7527 \global\bbl@divisibletrue
7528 \else
7529 \global\bbl@divisiblefalse
7530 \fi}}
7531 \newif\ifbbl@gregleap
7532 \def\bbl@ifgregleap#1{%
7533 \bbl@checkifdivisible{#1}{4}%

```



```

7534 \ifbbl@divisible
7535     \bbl@checkifdivisible{#1}{100}%
7536     \ifbbl@divisible
7537         \bbl@checkifdivisible{#1}{400}%
7538     \ifbbl@divisible
7539         \bbl@gregleaptrue
7540     \else
7541         \bbl@gregleapfalse
7542     \fi
7543 \else
7544     \bbl@gregleaptrue
7545 \fi
7546 \else
7547     \bbl@gregleapfalse
7548 \fi
7549 \ifbbl@gregleap}
7550 \def\bbl@gregdayspriormonths#1#2#3{%
7551     {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
7552         181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
7553     \bbl@ifgregleap{#2}%
7554     \ifnum #1 > 2
7555         \advance #3 by 1
7556     \fi
7557 \fi
7558 \global\bbl@cntcommon=#3}%
7559 #3=\bbl@cntcommon}
7560 \def\bbl@gregdaysprioryears#1#2{%
7561     {\countdef\tmpc=4
7562     \countdef\tmpb=2
7563     \tmpb=#1\relax
7564     \advance \tmpb by -1
7565     \tmpc=\tmpb
7566     \multiply \tmpc by 365
7567     #2=\tmpc
7568     \tmpc=\tmpb
7569     \divide \tmpc by 4
7570     \advance #2 by \tmpc
7571     \tmpc=\tmpb
7572     \divide \tmpc by 100
7573     \advance #2 by -\tmpc
7574     \tmpc=\tmpb
7575     \divide \tmpc by 400
7576     \advance #2 by \tmpc
7577     \global\bbl@cntcommon=#2\relax}%
7578 #2=\bbl@cntcommon}
7579 \def\bbl@absfromgreg#1#2#3#4{%
7580     {\countdef\tmpd=0
7581     #4=#1\relax
7582     \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
7583     \advance #4 by \tmpd
7584     \bbl@gregdaysprioryears{#3}{\tmpd}%
7585     \advance #4 by \tmpd
7586     \global\bbl@cntcommon=#4\relax}%
7587 #4=\bbl@cntcommon}
7588 \newif\ifbbl@hebrleap
7589 \def\bbl@checkleaphebryear#1{%
7590     {\countdef\tmpa=0
7591     \countdef\tmpb=1
7592     \tmpa=#1\relax
7593     \multiply \tmpa by 7
7594     \advance \tmpa by 1
7595     \bbl@remainder{\tmpa}{19}{\tmpb}%
7596     \ifnum \tmpb < 7

```

```

7597     \global\bbl@hebrleaptrue
7598 \else
7599     \global\bbl@hebrleapfalse
7600 \fi}}
7601 \def\bbl@hebreleapsedmonths#1#2{%
7602   {\countdef\tmpa=0
7603     \countdef\tmpb=1
7604     \countdef\tmpc=2
7605     \tmpa=#1\relax
7606     \advance \tmpa by -1
7607     #2=\tmpa
7608     \divide #2 by 19
7609     \multiply #2 by 235
7610     \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
7611     \tmpc=\tmpb
7612     \multiply \tmpb by 12
7613     \advance #2 by \tmpb
7614     \multiply \tmpc by 7
7615     \advance \tmpc by 1
7616     \divide \tmpc by 19
7617     \advance #2 by \tmpc
7618     \global\bbl@cntcommon=#2}%
7619   #2=\bbl@cntcommon}
7620 \def\bbl@hebreleapseddays#1#2{%
7621   {\countdef\tmpa=0
7622     \countdef\tmpb=1
7623     \countdef\tmpc=2
7624     \bbl@hebreleapsedmonths{#1}{#2}%
7625     \tmpa=#2\relax
7626     \multiply \tmpa by 13753
7627     \advance \tmpa by 5604
7628     \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
7629     \divide \tmpa by 25920
7630     \multiply #2 by 29
7631     \advance #2 by 1
7632     \advance #2 by \tmpa
7633     \bbl@remainder{#2}{7}{\tmpa}%
7634     \ifnum \tmpc < 19440
7635       \ifnum \tmpc < 9924
7636         \else
7637           \ifnum \tmpa=2
7638             \bbl@checkleaphebyear{#1}% of a common year
7639             \ifbbl@hebrleap
7640               \else
7641                 \advance #2 by 1
7642               \fi
7643             \fi
7644           \ifnum \tmpc < 16789
7645             \else
7646               \ifnum \tmpa=1
7647                 \advance #1 by -1
7648                 \bbl@checkleaphebyear{#1}% at the end of leap year
7649                 \ifbbl@hebrleap
7650                   \advance #2 by 1
7651                 \fi
7652               \fi
7653             \fi
7654           \fi
7655         \else
7656           \advance #2 by 1
7657         \fi
7658       \bbl@remainder{#2}{7}{\tmpa}%
7659       \ifnum \tmpa=0

```

```

7660     \advance #2 by 1
7661 \else
7662     \ifnum \tmpa=3
7663         \advance #2 by 1
7664     \else
7665         \ifnum \tmpa=5
7666             \advance #2 by 1
7667         \fi
7668     \fi
7669 \fi
7670 \global\bbl@cntcommon=#2\relax}%
7671 #2=\bbl@cntcommon}
7672 \def\bbl@daysinhebrewyear#1#2{%
7673 {\countdef\tmpe=12
7674  \bbl@hebreleapseddays{#1}{\tmpe}%
7675  \advance #1 by 1
7676  \bbl@hebreleapseddays{#1}{#2}%
7677  \advance #2 by -\tmpe
7678  \global\bbl@cntcommon=#2}%
7679 #2=\bbl@cntcommon}
7680 \def\bbl@hebrdayspriormonths#1#2#3{%
7681 {\countdef\tmpf= 14
7682  #3=\ifcase #1\relax
7683      0 \or
7684      0 \or
7685      30 \or
7686      59 \or
7687      89 \or
7688      118 \or
7689      148 \or
7690      148 \or
7691      177 \or
7692      207 \or
7693      236 \or
7694      266 \or
7695      295 \or
7696      325 \or
7697      400
7698  \fi
7699  \bbl@checkleaphebrewyear{#2}%
7700  \ifbbl@hebrleap
7701      \ifnum #1 > 6
7702          \advance #3 by 30
7703      \fi
7704  \fi
7705  \bbl@daysinhebrewyear{#2}{\tmpf}%
7706  \ifnum #1 > 3
7707      \ifnum \tmpf=353
7708          \advance #3 by -1
7709      \fi
7710      \ifnum \tmpf=383
7711          \advance #3 by -1
7712      \fi
7713  \fi
7714  \ifnum #1 > 2
7715      \ifnum \tmpf=355
7716          \advance #3 by 1
7717      \fi
7718      \ifnum \tmpf=385
7719          \advance #3 by 1
7720      \fi
7721  \fi
7722  \global\bbl@cntcommon=#3\relax}%

```

```

7723 #3=\bbl@cntcommon}
7724 \def\bbl@absfromhebr#1#2#3#4{%
7725   {#4=#1\relax
7726     \bbl@hebrdayspriormonths{#2}{#3}{#1}%
7727     \advance #4 by #1\relax
7728     \bbl@hebreleapseddays{#3}{#1}%
7729     \advance #4 by #1\relax
7730     \advance #4 by -1373429
7731     \global\bbl@cntcommon=#4\relax}%
7732 #4=\bbl@cntcommon}
7733 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
7734   {\countdef\tmpx= 17
7735     \countdef\tmpy= 18
7736     \countdef\tmpz= 19
7737     #6=#3\relax
7738     \global\advance #6 by 3761
7739     \bbl@absfromgreg{#1}{#2}{#3}{#4}%
7740     \tmpz=1 \tmpy=1
7741     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7742     \ifnum \tmpx > #4\relax
7743       \global\advance #6 by -1
7744       \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7745     \fi
7746     \advance #4 by -\tmpx
7747     \advance #4 by 1
7748     #5=#4\relax
7749     \divide #5 by 30
7750     \loop
7751       \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
7752       \ifnum \tmpx < #4\relax
7753         \advance #5 by 1
7754         \tmpy=\tmpx
7755       \repeat
7756       \global\advance #5 by -1
7757       \global\advance #4 by -\tmpy}}
7758 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebyear
7759 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
7760 \def\bbl@ca@hebrew#1-#2-#3@@#4#5#6{%
7761   \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
7762   \bbl@hebrfromgreg
7763   {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
7764   {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebyear}%
7765   \edef#4{\the\bbl@hebyear}%
7766   \edef#5{\the\bbl@hebrmonth}%
7767   \edef#6{\the\bbl@hebrday}}
7768 \</ca-hebrew>

```

## 17 Persian

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

```

7769 \<ca-persian>
7770 \ExplSyntaxOn
7771 \<<Compute Julian day>>
7772 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
7773   2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
7774 \def\bbl@ca@persian#1-#2-#3@@#4#5#6{%
7775   \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
7776   \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051

```

```

7777 \bbl@afterfi\expandafter\@gobble
7778 \fi\fi
7779 { \bbl@error{Year~out~of~range}{The~allowed~range~is~2013-2050}}%
7780 \bbl@xin@{ \bbl@tempa}{ \bbl@cs@firstjal@xx}%
7781 \ifin@ \def\bbl@tempe{20} \else \def\bbl@tempe{21} \fi
7782 \edef\bbl@tempc{ \fp_eval:n{ \bbl@cs@jd{ \bbl@tempa}{#2}{#3}+.5}}% current
7783 \edef\bbl@tempb{ \fp_eval:n{ \bbl@cs@jd{ \bbl@tempa}{03}{ \bbl@tempe}+.5}}% begin
7784 \ifnum\bbl@tempc<\bbl@tempb
7785 \edef\bbl@tempa{ \fp_eval:n{ \bbl@tempa-1}}% go back 1 year and redo
7786 \bbl@xin@{ \bbl@tempa}{ \bbl@cs@firstjal@xx}%
7787 \ifin@ \def\bbl@tempe{20} \else \def\bbl@tempe{21} \fi
7788 \edef\bbl@tempb{ \fp_eval:n{ \bbl@cs@jd{ \bbl@tempa}{03}{ \bbl@tempe}+.5}}%
7789 \fi
7790 \edef#4{ \fp_eval:n{ \bbl@tempa-621}}% set Jalali year
7791 \edef#6{ \fp_eval:n{ \bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
7792 \edef#5{ \fp_eval:n{ % set Jalali month
7793 (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
7794 \edef#6{ \fp_eval:n{ % set Jalali day
7795 (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6)))}}
7796 \ExplSyntaxOff
7797 </ca-persian>

```

## 18 Coptic and Ethiopic

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

```

7798 <*ca-coptic>
7799 \ExplSyntaxOn
7800 <<Compute Julian day>>
7801 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
7802 \edef\bbl@tempd{ \fp_eval:n{ floor( \bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
7803 \edef\bbl@tempc{ \fp_eval:n{ \bbl@tempd - 1825029.5}}%
7804 \edef#4{ \fp_eval:n{ %
7805 floor(( \bbl@tempc - floor(( \bbl@tempc+366) / 1461)) / 365) + 1}}%
7806 \edef\bbl@tempc{ \fp_eval:n{ %
7807 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
7808 \edef#5{ \fp_eval:n{ floor( \bbl@tempc / 30) + 1}}%
7809 \edef#6{ \fp_eval:n{ \bbl@tempc - (#5 - 1) * 30 + 1}}%
7810 \ExplSyntaxOff
7811 </ca-coptic>
7812 <*ca-ethiopic>
7813 \ExplSyntaxOn
7814 <<Compute Julian day>>
7815 \def\bbl@ca@ethiopic#1-#2-#3\@#4#5#6{%
7816 \edef\bbl@tempd{ \fp_eval:n{ floor( \bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
7817 \edef\bbl@tempc{ \fp_eval:n{ \bbl@tempd - 1724220.5}}%
7818 \edef#4{ \fp_eval:n{ %
7819 floor(( \bbl@tempc - floor(( \bbl@tempc+366) / 1461)) / 365) + 1}}%
7820 \edef\bbl@tempc{ \fp_eval:n{ %
7821 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
7822 \edef#5{ \fp_eval:n{ floor( \bbl@tempc / 30) + 1}}%
7823 \edef#6{ \fp_eval:n{ \bbl@tempc - (#5 - 1) * 30 + 1}}%
7824 \ExplSyntaxOff
7825 </ca-ethiopic>

```

## 19 Buddhist

That's very simple.

```

7826 <*ca-buddhist>
7827 \def\bbl@ca@buddhist#1-#2-#3\@#4#5#6{%
7828 \edef#4{ \number\numexpr#1+543\relax}%

```

```

7829 \edef#5{#2}%
7830 \edef#6{#3}}
7831 </ca-buddhist>

```

## 20 Support for Plain T<sub>E</sub>X (plain.def)

### 20.1 Not renaming hyphen.tex

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based T<sub>E</sub>X-format. When asked he responded:

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

People can have a file `localhyphen.tex` or whatever they like, but they mustn’t diddle with `hyphen.tex` (or `plain.tex` except to preload additional fonts).

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

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

```

7832 <*bplain | blplain>
7833 \catcode`\{=1 % left brace is begin-group character
7834 \catcode`\}=2 % right brace is end-group character
7835 \catcode`\#=6 % hash mark is macro parameter character

```

If a file called `hyphen.cfg` can be found, we make sure that *it* will be read instead of the file `hyphen.tex`. We do this by first saving the original meaning of `\input` (and I use a one letter control sequence for that so as not to waste multi-letter control sequence on this in the format).

```

7836 \openin 0 hyphen.cfg
7837 \ifeof0
7838 \else
7839 \let\input

```

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

```

7840 \def\input #1 {%
7841 \let\input\input
7842 \a hyphen.cfg
7843 \let\input\input
7844 }
7845 \fi
7846 </bplain | blplain>

```

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

```

7847 <bplain>\a plain.tex
7848 <blplain>\a lplain.tex

```

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

```

7849 <bplain>\def\fmtname{babel-plain}
7850 <blplain>\def\fmtname{babel-lplain}

```

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

## 20.2 Emulating some $\text{\LaTeX}$ features

The file `babel.def` expects some definitions made in the  $\text{\LaTeX} 2_{\epsilon}$  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 an alternative mechanism is provided. For the moment, only `\babeloptionstrings` and `\babeloptionmath` are provided, which can be defined before loading `babel`. `\BabelModifiers` can be set too (but not sure it works).

```
7851 <<{*Emulate LaTeX}>> ≡
7852 \def\@empty{}
7853 \def\loadlocalcfg#1{%
7854   \openin0#1.cfg
7855   \ifeof0
7856     \closein0
7857   \else
7858     \closein0
7859     {\immediate\write16{*****}%
7860      \immediate\write16{* Local config file #1.cfg used}%
7861      \immediate\write16{*}%
7862     }
7863   \input #1.cfg\relax
7864 \fi
7865 \@endofldf}
```

## 20.3 General tools

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

```
7866 \long\def\@firstofone#1{#1}
7867 \long\def\@firstoftwo#1#2{#1}
7868 \long\def\@secondoftwo#1#2{#2}
7869 \def\@nnil{\@nil}
7870 \def\@gobbletwo#1#2{}
7871 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7872 \def\@star@or@long#1{%
7873   \@ifstar
7874   {\let\@ngrel@x\relax#1}%
7875   {\let\@ngrel@x\long#1}}
7876 \let\@ngrel@x\relax
7877 \def\@car#1#2\@nil{#1}
7878 \def\@cdr#1#2\@nil{#2}
7879 \let\@typeset@protect\relax
7880 \let\protected@edef\edef
7881 \long\def\@gobble#1{}
7882 \edef\@backslashchar{\expandafter\@gobble\string\}
7883 \def\strip@prefix#1>{}
7884 \def\g@addto@macro#1#2{%
7885   \toks@\expandafter{#1#2}%
7886   \xdef#1{\the\toks@}}
7887 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
7888 \def\@nameuse#1{\csname #1\endcsname}
7889 \def\@ifundefined#1{%
7890   \expandafter\ifx\csname#1\endcsname\relax
7891     \expandafter\@firstoftwo
7892   \else
7893     \expandafter\@secondoftwo
7894   \fi}
7895 \def\@expandtwoargs#1#2#3{%
7896   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7897 \def\zap@space#1 #2{%
7898   #1%
7899   \ifx#2\@empty\else\expandafter\zap@space\fi
7900   #2}
7901 \let\bbl@trace\@gobble
7902 \def\bbl@error#1#2{%
```

```

7903 \begingroup
7904 \newlinechar=`^^J
7905 \def\{^^J(babel) }%
7906 \errhelp{#2}\errmessage{\#1}%
7907 \endgroup}
7908 \def\bbl@warning#1{%
7909 \begingroup
7910 \newlinechar=`^^J
7911 \def\{^^J(babel) }%
7912 \message{\#1}%
7913 \endgroup}
7914 \let\bbl@infowarn\bbl@warning
7915 \def\bbl@info#1{%
7916 \begingroup
7917 \newlinechar=`^^J
7918 \def\{^^J}%
7919 \wlog{#1}%
7920 \endgroup}

```

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

```

7921 \ifx\@preamblecmds\@undefined
7922 \def\@preamblecmds{}
7923 \fi
7924 \def\@onlypreamble#1{%
7925 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
7926 \@preamblecmds\do#1}}
7927 \@onlypreamble\@onlypreamble

```

Mimick  $\TeX$ 's `\AtBeginDocument`; for this to work the user needs to add `\begindocument` to his file.

```

7928 \def\begindocument{%
7929 \@begindocumenthook
7930 \global\let\@begindocumenthook\@undefined
7931 \def\do##1{\global\let##1\@undefined}%
7932 \@preamblecmds
7933 \global\let\do\noexpand}
7934 \ifx\@begindocumenthook\@undefined
7935 \def\@begindocumenthook{}
7936 \fi
7937 \@onlypreamble\@begindocumenthook
7938 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

We also have to mimick  $\TeX$ 's `\AtEndOfPackage`. Our replacement macro is much simpler; it stores its argument in `\@endoflfd`.

```

7939 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
7940 \@onlypreamble\AtEndOfPackage
7941 \def\@endoflfd{}
7942 \@onlypreamble\@endoflfd
7943 \let\bbl@afterlang\@empty
7944 \chardef\bbl@opt@hyphenmap\z@

```

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

```

7945 \catcode`\&=\z@
7946 \ifx&\if@files\@undefined
7947 \expandafter\let\csname if@files\endcsname
7948 \csname iffalse\endcsname
7949 \fi
7950 \catcode`\&=4

```

Mimick  $\TeX$ 's commands to define control sequences.

```

7951 \def\newcommand{\@star@or@long\new@command}

```



```

7952 \def\new@command#1{%
7953   \@testopt{\@newcommand#1}0}
7954 \def\@newcommand#1[#2]{%
7955   \@ifnextchar [{\@xargdef#1[#2]}%
7956                 {\@argdef#1[#2]}}
7957 \long\def\@argdef#1[#2]#3{%
7958   \@yargdef#1\@ne{#2}{#3}}
7959 \long\def\@xargdef#1[#2][#3]#4{%
7960   \expandafter\def\expandafter#1\expandafter{%
7961     \expandafter\@protected@testopt\expandafter #1%
7962     \csname\string#1\expandafter\endcsname{#3}}%
7963   \expandafter\@yargdef \csname\string#1\endcsname
7964   \tw@{#2}{#4}}
7965 \long\def\@yargdef#1#2#3{%
7966   \@tempcnta#3\relax
7967   \advance \@tempcnta \@ne
7968   \let\@hash@\relax
7969   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7970   \@tempcntb #2%
7971   \@whilenum \@tempcntb < \@tempcnta
7972   \do{%
7973     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
7974     \advance\@tempcntb \@ne}%
7975   \let\@hash@##%
7976   \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
7977 \def\providecommand{\@star@or@long\provide@command}
7978 \def\provide@command#1{%
7979   \begingroup
7980     \escapechar\m@ne\def\@gtempa{{\string#1}}%
7981   \endgroup
7982   \expandafter\ifundefined\@gtempa
7983     {\def\reserved@a{\new@command#1}}%
7984     {\let\reserved@a\relax
7985      \def\reserved@a{\new@command\reserved@a}}%
7986   \reserved@a}%
7987 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7988 \def\declare@robustcommand#1{%
7989   \edef\reserved@a{\string#1}%
7990   \def\reserved@b{#1}%
7991   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7992   \edef#1{%
7993     \ifx\reserved@a\reserved@b
7994       \noexpand\x@protect
7995       \noexpand#1%
7996     \fi
7997     \noexpand\protect
7998     \expandafter\noexpand\csname
7999       \expandafter\@gobble\string#1 \endcsname
8000   }%
8001   \expandafter\new@command\csname
8002     \expandafter\@gobble\string#1 \endcsname
8003 }
8004 \def\x@protect#1{%
8005   \ifx\protect\@typeset@protect\else
8006     \@x@protect#1%
8007   \fi
8008 }
8009 \catcode\&=\z@ % Trick to hide conditionals
8010 \def\@x@protect#1&fi#2#3{&fi\protect#1}

```

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

```

8011 \def\bbl@tempa{\csname newif\endcsname&ifin@}
8012 \catcode`\&=4
8013 \ifx\in@\undefined
8014 \def\in@#1#2{%
8015 \def\in@@#1#1##2##3\in@{%
8016 \ifx\in@@#2\in@false\else\in@true\fi}%
8017 \in@@#2#1\in@\in@}
8018 \else
8019 \let\bbl@tempa\empty
8020 \fi
8021 \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).

```

8022 \def\@ifpackagewith#1#2#3#4{#3}

```

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

```

8023 \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  $\LaTeX 2_{\epsilon}$  versions; just enough to make things work in plain  $\TeX$  environments.

```

8024 \ifx\@tempcnta\undefined
8025 \csname newcount\endcsname\@tempcnta\relax
8026 \fi
8027 \ifx\@tempcntb\undefined
8028 \csname newcount\endcsname\@tempcntb\relax
8029 \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`).

```

8030 \ifx\bye\undefined
8031 \advance\count10 by -2\relax
8032 \fi
8033 \ifx\@ifnextchar\undefined
8034 \def\@ifnextchar#1#2#3{%
8035 \let\reserved@d=#1%
8036 \def\reserved@a{#2}\def\reserved@b{#3}%
8037 \futurelet\@let@token\@ifnch}
8038 \def\@ifnch{%
8039 \ifx\@let@token\sptoken
8040 \let\reserved@c\@xifnch
8041 \else
8042 \ifx\@let@token\reserved@d
8043 \let\reserved@c\reserved@a
8044 \else
8045 \let\reserved@c\reserved@b
8046 \fi
8047 \fi
8048 \reserved@c}
8049 \def\:{\let\sptoken= }\: % this makes \sptoken a space token
8050 \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
8051 \fi
8052 \def\@testopt#1#2{%
8053 \@ifnextchar[#{1}{#1[#2]}}
8054 \def\@protected@testopt#1{%
8055 \ifx\protect\@typeset@protect
8056 \expandafter\@testopt
8057 \else
8058 \@x@protect#1%

```

```

8059 \fi}
8060 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@whilenum{#1\relax
8061 #2\relax}\fi}
8062 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8063 \else\expandafter\@gobble\fi{#1}}

```

## 20.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain  $\TeX$  environment.

```

8064 \def\DeclareTextCommand{%
8065   \@dec@text@cmd\providecommand
8066 }
8067 \def\ProvideTextCommand{%
8068   \@dec@text@cmd\providecommand
8069 }
8070 \def\DeclareTextSymbol#1#2#3{%
8071   \@dec@text@cmd\chardef#1{#2}#3\relax
8072 }
8073 \def\@dec@text@cmd#1#2#3{%
8074   \expandafter\def\expandafter#2%
8075     \expandafter{%
8076       \csname#3-cmd\expandafter\endcsname
8077       \expandafter#2%
8078       \csname#3\string#2\endcsname
8079     }%
8080 %   \let\@ifdefinable\@rc@ifdefinable
8081   \expandafter#1\csname#3\string#2\endcsname
8082 }
8083 \def\@current@cmd#1{%
8084   \ifx\protect\@typeset@protect\else
8085     \noexpand#1\expandafter\@gobble
8086   \fi
8087 }
8088 \def\@changed@cmd#1#2{%
8089   \ifx\protect\@typeset@protect
8090     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8091       \expandafter\ifx\csname ?\string#1\endcsname\relax
8092         \expandafter\def\csname ?\string#1\endcsname{%
8093           \@changed@x@err{#1}%
8094         }%
8095       \fi
8096       \global\expandafter\let
8097         \csname\cf@encoding\string#1\expandafter\endcsname
8098         \csname ?\string#1\endcsname
8099       \fi
8100     \csname\cf@encoding\string#1%
8101     \expandafter\endcsname
8102   \else
8103     \noexpand#1%
8104   \fi
8105 }
8106 \def\@changed@x@err#1{%
8107   \errhelp{Your command will be ignored, type <return> to proceed}%
8108   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8109 \def\DeclareTextCommandDefault#1{%
8110   \DeclareTextCommand#1?%
8111 }
8112 \def\ProvideTextCommandDefault#1{%
8113   \ProvideTextCommand#1?%
8114 }
8115 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8116 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8117 \def\DeclareTextAccent#1#2#3{%

```

```

8118 \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
8119 }
8120 \def\DeclareTextCompositeCommand#1#2#3#4{%
8121   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8122   \edef\reserved@b{\string##1}%
8123   \edef\reserved@c{%
8124     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8125   \ifx\reserved@b\reserved@c
8126     \expandafter\expandafter\expandafter\ifx
8127       \expandafter\@car\reserved@a\relax\relax\@nil
8128       \@text@composite
8129     \else
8130       \edef\reserved@b##1{%
8131         \def\expandafter\noexpand
8132           \csname#2\string#1\endcsname####1{%
8133             \noexpand\@text@composite
8134             \expandafter\noexpand\csname#2\string#1\endcsname
8135             ####1\noexpand\@empty\noexpand\@text@composite
8136             {##1}%
8137           }%
8138         }%
8139       \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8140     \fi
8141     \expandafter\def\csname\expandafter\string\csname
8142       #2\endcsname\string#1-\string#3\endcsname{#4}
8143   \else
8144     \errhelp{Your command will be ignored, type <return> to proceed}%
8145     \errmessage{\string\DeclareTextCompositeCommand\space used on
8146       inappropriate command \protect#1}
8147   \fi
8148 }
8149 \def\@text@composite#1#2#3\@text@composite{%
8150   \expandafter\@text@composite@x
8151     \csname\string#1-\string#2\endcsname
8152 }
8153 \def\@text@composite@x#1#2{%
8154   \ifx#1\relax
8155     #2%
8156   \else
8157     #1%
8158   \fi
8159 }
8160 %
8161 \def\@strip@args#1:#2-#3\@strip@args{#2}
8162 \def\DeclareTextComposite#1#2#3#4{%
8163   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8164   \bgroup
8165     \lccode`\@=#4%
8166     \lowercase{%
8167       \egroup
8168       \reserved@a @%
8169     }%
8170 }
8171 %
8172 \def\UseTextSymbol#1#2{#2}
8173 \def\UseTextAccent#1#2#3{}
8174 \def\@use@text@encoding#1{}
8175 \def\DeclareTextSymbolDefault#1#2{%
8176   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8177 }
8178 \def\DeclareTextAccentDefault#1#2{%
8179   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8180 }

```

```
8181 \def\cf@encoding{OT1}
```

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

```
8182 \DeclareTextAccent{"}{OT1}{127}
8183 \DeclareTextAccent{'}{OT1}{19}
8184 \DeclareTextAccent{^}{OT1}{94}
8185 \DeclareTextAccent`}{OT1}{18}
8186 \DeclareTextAccent~}{OT1}{126}
```

The following control sequences are used in `babel.def` but are not defined for PLAIN  $\text{\TeX}$ .

```
8187 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8188 \DeclareTextSymbol{\textquotedblright}{OT1}{`"}
8189 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
8190 \DeclareTextSymbol{\textquoteright}{OT1}{``'}
8191 \DeclareTextSymbol{\i}{OT1}{16}
8192 \DeclareTextSymbol{\ss}{OT1}{25}
```

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

```
8193 \ifx\scriptsize\@undefined
8194   \let\scriptsize\sevenrm
8195 \fi
```

And a few more “dummy” definitions.

```
8196 \def\languagename{english}%
8197 \let\bbl@opt@shorthands\@nnil
8198 \def\bbl@ifshorthand#1#2#3#2}%
8199 \let\bbl@language@opts\@empty
8200 \ifx\babeloptionstrings\@undefined
8201   \let\bbl@opt@strings\@nnil
8202 \else
8203   \let\bbl@opt@strings\babeloptionstrings
8204 \fi
8205 \def\BabelStringsDefault{generic}
8206 \def\bbl@tempa{normal}
8207 \ifx\babeloptionmath\bbl@tempa
8208   \def\bbl@mathnormal{\noexpand\textormath}
8209 \fi
8210 \def\AfterBabelLanguage#1#2{}
8211 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
8212 \let\bbl@afterlang\relax
8213 \def\bbl@opt@safe{BR}
8214 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
8215 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
8216 \expandafter\newif\csname ifbbl@single\endcsname
8217 \chardef\bbl@bidimode\z@
8218 <</Emulate LaTeX>>
```

A proxy file:

```
8219 <*plain>
8220 \input babel.def
8221 </plain>
```

## 21 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

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national  $\text{\LaTeX}$  styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The  $\text{\TeX}$ book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport,  *$\text{\LaTeX}$ , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in:  $\text{\TeX}$ hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018
- [10] Hubert Partl, *German  $\text{\TeX}$* , *TUGboat* 9 (1988) #1, p. 70–72.
- [11] Joachim Schrod, *International  $\text{\LaTeX}$  is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using  $\text{\LaTeX}$* , Springer, 2002, p. 301–373.
- [13] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).