

# 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 3.83.2938  
2022/12/01

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 . . . . .	27
1.16	Creating a language . . . . .	28
1.17	Digits and counters . . . . .	32
1.18	Dates . . . . .	33
1.19	Accessing language info . . . . .	34
1.20	Hyphenation and line breaking . . . . .	35
1.21	Transforms . . . . .	37
1.22	Selection based on BCP 47 tags . . . . .	40
1.23	Selecting scripts . . . . .	41
1.24	Selecting directions . . . . .	42
1.25	Language attributes . . . . .	45
1.26	Hooks . . . . .	46
1.27	Languages supported by babel with ldf files . . . . .	47
1.28	Unicode character properties in luatex . . . . .	48
1.29	Tweaking some features . . . . .	49
1.30	Tips, workarounds, known issues and notes . . . . .	49
1.31	Current and future work . . . . .	50
1.32	Tentative and experimental code . . . . .	50
<b>2</b>	<b>Loading languages with language.dat</b>	<b>51</b>
2.1	Format . . . . .	51
<b>3</b>	<b>The interface between the core of babel and the language definition files</b>	<b>52</b>
3.1	Guidelines for contributed languages . . . . .	53
3.2	Basic macros . . . . .	53
3.3	Skeleton . . . . .	55
3.4	Support for active characters . . . . .	56
3.5	Support for saving macro definitions . . . . .	56
3.6	Support for extending macros . . . . .	56
3.7	Macros common to a number of languages . . . . .	56
3.8	Encoding-dependent strings . . . . .	57
3.9	Executing code based on the selector . . . . .	60
<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>61</b>

<b>6</b>	<b>Tools</b>	<b>61</b>
6.1	Multiple languages	66
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	68
6.5	Conditional loading of shorthands	69
6.6	Interlude for Plain	71
<b>7</b>	<b>Multiple languages</b>	<b>71</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	87
7.6	Language attributes	96
7.7	Support for saving macro definitions	98
7.8	Short tags	99
7.9	Hyphens	99
7.10	Multiencoding strings	101
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	109
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>135</b>
8.1	Cross referencing macros	137
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>	141
8.3.3	<code>hhline</code>	141
8.4	Encoding and fonts	142
8.5	Basic bidi support	144
8.6	Local Language Configuration	147
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>151</b>
<b>11</b>	<b>Font handling with <code>fontspec</code></b>	<b>155</b>
<b>12</b>	<b>Hooks for XeTeX and LuaTeX</b>	<b>158</b>
12.1	XeTeX	158
12.2	Layout	160
12.3	LuaTeX	161
12.4	Southeast Asian scripts	167
12.5	CJK line breaking	169
12.6	Arabic justification	171
12.7	Common stuff	175
12.8	Automatic fonts and ids switching	175
12.9	Bidi	179
12.10	Layout	181
12.11	Lua: transforms	187

12.12	Lua: Auto bidi with basic and basic-r . . . . .	195
<b>13</b>	<b>Data for CJK</b>	<b>205</b>
<b>14</b>	<b>The ‘nil’ language</b>	<b>205</b>
<b>15</b>	<b>Calendars</b>	<b>206</b>
15.1	Islamic . . . . .	206
<b>16</b>	<b>Hebrew</b>	<b>208</b>
<b>17</b>	<b>Persian</b>	<b>212</b>
<b>18</b>	<b>Coptic and Ethiopic</b>	<b>213</b>
<b>19</b>	<b>Buddhist</b>	<b>213</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 . . . . .	215
20.4	Encoding related macros . . . . .	218
<b>21</b>	<b>Acknowledgements</b>	<b>221</b>

## Troubleshooting

Paragraph ended before \UTFviii@three@octets was complete . . . . .	5
No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the format . . . . .	5
You are loading directly a language style . . . . .	8
Unknown language ‘LANG’ . . . . .	8
Argument of \language@active@arg” has an extra } . . . . .	12
Package 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  $\text{\LaTeX}$ , the preamble of the document:

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

would tell  $\text{\LaTeX}$  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 `khh` 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 \knbcode, 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` {<shorthands-list>}  
`\shorthandoff` \*{<shorthands-list>}

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

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*`{*<char>*} is provided, which makes sure shorthands are always activated.

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

## `\defineshortand` [*<language>* , *<language>* , ... ] {*<shorthand>* } {*<code>* }

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`{*<lang>*} to the corresponding `\extras`{*<lang>*}, as explained below). By default, user shorthands are (re)defined.

User shorthands override language ones, which in turn override system shorthands.

Language-dependent user shorthands (new in 3.9) take precedence over “normal” user shorthands.

**EXAMPLE** Let’s assume you want a unified set of shorthand for 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` {*<language>*}

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}\dots}
```

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  $\text{\LaTeX}$ , an alternative method to execute some code just after an `ldf` file is loaded is with `\AddToHook` and the hook file `<language>.ldf/after`. `Babel` does not predeclare it, and you have to do it yourself with `\ActivateGenericHook`.

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

## 1.13 ini files

An alternative approach to define a language (or, more precisely, a *locale*) is by means of an `ini` file. Currently `babel` provides about 250 of these files containing the basic data required for a locale, plus basic templates for 500 about locales.

`ini` files are not meant only for `babel`, and they 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 `\dotsname` 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-DZ	Arabic <sup>u</sup>	bm	Bambara
ar-EG	Arabic <sup>u</sup>	bn	Bangla <sup>u</sup>
ar-IQ	Arabic <sup>u</sup>	bo	Tibetan <sup>u</sup>
ar-JO	Arabic <sup>u</sup>	br	Breton <sup>ul</sup>
ar-LB	Arabic <sup>u</sup>	brx	Bodo
ar-MA	Arabic <sup>u</sup>	bs-Cyrl	Bosnian
ar-PS	Arabic <sup>u</sup>	bs-Latn	Bosnian <sup>ul</sup>
ar-SA	Arabic <sup>u</sup>	bs	Bosnian <sup>ul</sup>
ar-SY	Arabic <sup>u</sup>	ca	Catalan <sup>ul</sup>
ar-TN	Arabic <sup>u</sup>	ce	Chechen
ar	Arabic <sup>u</sup>	cgg	Chiga
as	Assamese <sup>u</sup>	chr	Cherokee
asa	Asu	ckb-Arab	Central Kurdish <sup>u</sup>
ast	Asturian <sup>ul</sup>	ckb-Latn	Central Kurdish <sup>u</sup>
az-Cyrl	Azerbaijani	ckb	Central Kurdish <sup>u</sup>
az-Latn	Azerbaijani	cop	Coptic
az	Azerbaijani <sup>ul</sup>	cs	Czech <sup>ul</sup>
bas	Basaa	cu-Cyrs	Church Slavic <sup>u</sup>

cu-Glag	Church Slavic	haw	Hawaiian
cu	Church Slavic <sup>u</sup>	he	Hebrew <sup>ul</sup>
cy	Welsh <sup>ul</sup>	hi	Hindi <sup>u</sup>
da	Danish <sup>ul</sup>	hr	Croatian <sup>ul</sup>
dav	Taita	hsb	Upper Sorbian <sup>ul</sup>
de-1901	German <sup>ul</sup>	hu	Hungarian <sup>ul</sup>
de-1996	German <sup>ul</sup>	hy	Armenian <sup>ul</sup>
de-AT-1901	Austrian German <sup>ul</sup>	ia	Interlingua <sup>ul</sup>
de-AT-1996	Austrian German <sup>ul</sup>	id	Indonesian <sup>ul</sup>
de-AT	Austrian German <sup>ul</sup>	ig	Igbo
de-CH-1901	Swiss High German <sup>ul</sup>	ii	Sichuan Yi
de-CH-1996	Swiss High 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>u</sup>
dyo	Jola-Fonyi	kab	Kabyle
dz	Dzongkha	kam	Kamba
ebu	Embu	kde	Makonde
ee	Ewe	kea	Kabuverdianu
el-polyton	Polytonic Greek <sup>ul</sup>	kgp	Kaingang
el	Greek <sup>ul</sup>	khq	Koyra Chiini
en-AU	Australian English <sup>ul</sup>	ki	Kikuyu
en-CA	Canadian English <sup>ul</sup>	kk	Kazakh
en-GB	British English <sup>ul</sup>	kkj	Kako
en-NZ	English <sup>ul</sup>	kl	Kalaallisut
en-US	American English <sup>ul</sup>	kln	Kalenjin
en	English <sup>ul</sup>	km	Khmer <sup>u</sup>
eo	Esperanto <sup>ul</sup>	kmr-Arab	Northern Kurdish <sup>u</sup>
es-MX	Mexican Spanish <sup>ul</sup>	kmr-Latn	Northern Kurdish <sup>ul</sup>
es	Spanish <sup>ul</sup>	kmr	Northern Kurdish <sup>ul</sup>
et	Estonian <sup>ul</sup>	kn	Kannada <sup>u</sup>
eu	Basque <sup>ul</sup>	ko-Hani	Korean <sup>u</sup>
ewo	Ewondo	ko	Korean <sup>u</sup>
fa	Persian <sup>u</sup>	kok	Konkani
ff	Fulah	ks	Kashmiri
fi	Finnish <sup>ul</sup>	ksb	Shambala
fil	Filipino	ksf	Bafia
fo	Faroese	ksh	Colognian
fr-BE	French <sup>ul</sup>	kw	Cornish
fr-CA	Canadian French <sup>ul</sup>	ky	Kyrgyz
fr-CH	Swiss French <sup>ul</sup>	la-x-classic	Classic Latin <sup>ul</sup>
fr-LU	French <sup>ul</sup>	la-x-ecclesia	Ecclesiastic Latin <sup>ul</sup>
fr	French <sup>ul</sup>	la-x-medieval	Medieval Latin <sup>ul</sup>
fur	Friulian <sup>ul</sup>	la	Latin <sup>ul</sup>
fy	Western Frisian	lag	Langi
ga	Irish <sup>ul</sup>	lb	Luxembourgish <sup>ul</sup>
gd	Scottish Gaelic <sup>ul</sup>	lg	Ganda
gl	Galician <sup>ul</sup>	lkt	Lakota
grc	Ancient Greek <sup>ul</sup>	ln	Lingala
gsw	Swiss German	lo	Lao <sup>u</sup>
gu	Gujarati	lrc	Northern Luri
guz	Gusii	lt	Lithuanian <sup>ul</sup>
gv	Manx	lu	Luba-Katanga
ha-GH	Hausa	luo	Luo
ha-NE	Hausa	luy	Luyia
ha	Hausa <sup>ul</sup>	lv	Latvian <sup>ul</sup>

mas	Masai	saq	Samburu
mer	Meru	sbp	Sangu
mfe	Morisyen	sc	Sardinian
mg	Malagasy	se	Northern Sami <sup>ul</sup>
mgh	Makhuwa-Meetto	seh	Sena
mgo	Meta'	ses	Koyraboro Senni
mk	Macedonian <sup>ul</sup>	sg	Sango
ml	Malayalam <sup>u</sup>	shi-Latn	Tachelhit
mn	Mongolian	shi-Tfng	Tachelhit
mr	Marathi <sup>u</sup>	shi	Tachelhit
ms-BN	Malay	si	Sinhala <sup>u</sup>
ms-SG	Malay	sk	Slovak <sup>ul</sup>
ms	Malay <sup>ul</sup>	sl	Slovenian <sup>ul</sup>
mt	Maltese	smn	Inari Sami
mua	Mundang	sn	Shona
my	Burmese	so	Somali
mzn	Mazanderani	sq	Albanian <sup>ul</sup>
naq	Nama	sr-Cyrl-BA	Serbian <sup>ul</sup>
nb	Norwegian Bokmål <sup>ul</sup>	sr-Cyrl-ME	Serbian <sup>ul</sup>
nd	North Ndebele	sr-Cyrl-XK	Serbian <sup>ul</sup>
ne	Nepali	sr-Cyrl	Serbian <sup>ul</sup>
nl	Dutch <sup>ul</sup>	sr-Latn-BA	Serbian <sup>ul</sup>
nmg	Kwasio	sr-Latn-ME	Serbian <sup>ul</sup>
nn	Norwegian Nynorsk <sup>ul</sup>	sr-Latn-XK	Serbian <sup>ul</sup>
nnh	Ngiemboon	sr-Latn	Serbian <sup>ul</sup>
no	Norwegian <sup>ul</sup>	sr	Serbian <sup>ul</sup>
nus	Nuer	sv	Swedish <sup>ul</sup>
nyn	Nyankole	sw	Swahili
oc	Occitan <sup>ul</sup>	syr	Syriac
om	Oromo	ta	Tamil <sup>u</sup>
or	Odia	te	Telugu <sup>u</sup>
os	Ossetic	teo	Teso
pa-Arab	Punjabi	th	Thai <sup>ul</sup>
pa-Guru	Punjabi <sup>u</sup>	ti	Tigrinya
pa	Punjabi <sup>u</sup>	tk	Turkmen <sup>ul</sup>
pl	Polish <sup>ul</sup>	to	Tongan
pms	Piedmontese <sup>ul</sup>	tr	Turkish <sup>ul</sup>
ps	Pashto	twq	Tasawaq
pt-BR	Brazilian Portuguese <sup>ul</sup>	tzm	Central Atlas Tamazight
pt-PT	European Portuguese <sup>ul</sup>	ug	Uyghur <sup>u</sup>
pt	Portuguese <sup>ul</sup>	uk	Ukrainian <sup>ul</sup>
qu	Quechua	ur	Urdu <sup>u</sup>
rm	Romansh <sup>ul</sup>	uz-Arab	Uzbek
rn	Rundi	uz-Cyrl	Uzbek
ro-MD	Moldavian <sup>ul</sup>	uz-Latn	Uzbek
ro	Romanian <sup>ul</sup>	uz	Uzbek
rof	Rombo	vai-Latn	Vai
ru	Russian <sup>ul</sup>	vai-Vaii	Vai
rw	Kinyarwanda	vai	Vai
rwk	Rwa	vi	Vietnamese <sup>ul</sup>
sa-Beng	Sanskrit	vun	Vunjo
sa-Deva	Sanskrit	wae	Walser
sa-Gujr	Sanskrit	xog	Soga
sa-Knda	Sanskrit	yav	Yangben
sa-Mlym	Sanskrit	yi	Yiddish
sa-Telu	Sanskrit	yo	Yoruba
sa	Sanskrit	yrl	Nheengatu
sah	Sakha	yue	Cantonese

zgh	Standard Moroccan Tamazight	zh-Hant-HK	Chinese
zh-Hans-HK	Chinese	zh-Hant-MO	Chinese
zh-Hans-MO	Chinese	zh-Hant	Chinese <sup>u</sup>
zh-Hans-SG	Chinese	zh	Chinese <sup>u</sup>
zh-Hans	Chinese <sup>u</sup>	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 `\babel provide` with a valueless `import`.

afrikaans	bulgarian
aghem	burmese
akan	canadian
albanian	cantonese
american	catalan
amharic	centralatlastamazight
ancientgreek	centralkurdish
arabic	chechen
arabic-algeria	cherokee
arabic-DZ	chiga
arabic-morocco	chinese-hans-hk
arabic-MA	chinese-hans-mo
arabic-syria	chinese-hans-sg
arabic-SY	chinese-hans
armenian	chinese-hant-hk
assamese	chinese-hant-mo
asturian	chinese-hant
asu	chinese-simplified-hongkongsarchina
australian	chinese-simplified-macausarchina
austrian	chinese-simplified-singapore
azerbaijani-cyrillic	chinese-simplified
azerbaijani-cyrl	chinese-traditional-hongkongsarchina
azerbaijani-latin	chinese-traditional-macausarchina
azerbaijani-latn	chinese-traditional
azerbaijani	chinese
bafia	churchslavic
bambara	churchslavic-cyrs
basaa	churchslavic-oldcyrillic <sup>12</sup>
basque	churchsslavic-glag
belarusian	churchsslavic-glagolitic
bemba	colognian
beni	cornish
bangla	croatian
bodo	czech
bosnian-cyrillic	danish
bosnian-cyrl	duala
bosnian-latin	dutch
bosnian-latn	dzongkha
bosnian	embu
brazilian	english-au
breton	english-australia
british	english-ca

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

english-canada  
english-gb  
english-newzealand  
english-nz  
english-unitedkingdom  
english-unitedstates  
english-us  
english  
esperanto  
estonian  
ewe  
ewondo  
faroese  
filipino  
finnish  
french-be  
french-belgium  
french-ca  
french-canada  
french-ch  
french-lu  
french-luxembourg  
french-switzerland  
french  
friulian  
fulah  
galician  
ganda  
georgian  
german-at  
german-austria  
german-ch  
german-switzerland  
german  
greek  
gujarati  
gusii  
hausa-gh  
hausa-ghana  
hausa-ne  
hausa-niger  
hausa  
hawaiian  
hebrew  
hindi  
hungarian  
icelandic  
igbo  
inarisami  
indonesian  
interlingua  
irish  
italian  
japanese  
jolafonyi  
kabuverdianu  
kabyle  
kako

kalaallisut  
kalenjin  
kamba  
kannada  
kashmiri  
kazakh  
khmer  
kikuyu  
kinyarwanda  
konkani  
korean  
koyraborosenni  
koyrachiini  
kwasio  
kyrgyz  
lakota  
langi  
lao  
latvian  
lingala  
lithuanian  
lowersorbian  
lsorbian  
lubakatanga  
luo  
luxembourgish  
luyia  
macedonian  
machame  
makhuwameetto  
makonde  
malagasy  
malay-bn  
malay-brunei  
malay-sg  
malay-singapore  
malay  
malayalam  
maltese  
manx  
marathi  
masai  
mazanderani  
meru  
meta  
mexican  
mongolian  
morisyen  
mundang  
nama  
nepali  
newzealand  
ngiemboon  
ngomba  
norsk  
northernluri  
northernsami  
northndebele



norwegianbokmal	serbian-cyrl-xk
norwegiannynorsk	serbian-cyrl
nswissgerman	serbian-latin-bosniaherzegovina
nuer	serbian-latin-kosovo
nyankole	serbian-latin-montenegro
nynorsk	serbian-latin
occitan	serbian-latn-ba
oriya	serbian-latn-me
oromo	serbian-latn-xk
ossetic	serbian-latn
pashto	serbian
persian	shambala
piedmontese	shona
polish	sichuanyi
polytonicgreek	sinhala
portuguese-br	slovak
portuguese-brazil	slovene
portuguese-portugal	slovenian
portuguese-pt	soga
portuguese	somali
punjabi-arab	spanish-mexico
punjabi-arabic	spanish-mx
punjabi-gurmukhi	spanish
punjabi-guru	standardmoroccantamazight
punjabi	swahili
quechua	swedish
romanian	swissgerman
romansh	tachelhit-latin
rombo	tachelhit-latn
rundi	tachelhit-tfng
russian	tachelhit-tifinagh
rwa	tachelhit
sakha	taita
samburu	tamil
samin	tasawaq
sango	telugu
sangu	teso
sanskrit-beng	thai
sanskrit-bengali	tibetan
sanskrit-deva	tigrinya
sanskrit-devanagari	tongan
sanskrit-gujarati	turkish
sanskrit-gujr	turkmen
sanskrit-kannada	ukenglish
sanskrit-knda	ukrainian
sanskrit-malayalam	upporsorbian
sanskrit-mlym	urdu
sanskrit-telu	usenglish
sanskrit-telugu	usorbian
sanskrit	uyghur
scottishgaelic	uzbek-arab
sena	uzbek-arabic
serbian-cyrillic-bosniaherzegovina	uzbek-cyrillic
serbian-cyrillic-kosovo	uzbek-cyrl
serbian-cyrillic-montenegro	uzbek-latin
serbian-cyrillic	uzbek-latn
serbian-cyrl-ba	uzbek
serbian-cyrl-me	vai-latin

vai-latn	welsh
vai-vai	westernfrisian
vai-vaii	yangben
vai	yiddish
vietnam	yoruba
vietnamese	zarma
vunjo	zulu
walser	

### 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 `\babel font`.<sup>13</sup>

`\babel font` [*<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 `\babel font` 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, `\babel font{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 `\babel font` 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}

\babel font{rm}{FreeSerif}
```

<sup>13</sup>See also the package `combofont` for a complementary approach.

```

\begin{document}

Svenska \foreignlanguage{hebrew}{עברית} 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}

```

`\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 imported 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\captionenglish{%
  \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 added to `\extras<lang>`:

```
\addto\extrasrussian{\mymacro}
```

There is a counterpart for code to be run when a language is unselected: `\noextras⟨lang⟩`.

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

Another way to modify a language loaded as a package or class option is by means of `\babelprovide`, described below in depth. So, something like:

```
\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  $\TeX$  sense). It only makes sense as the last value (or the only one; the subsequent ones are silently ignored). It is mostly useful with `luatex`, because you can add some patterns with `\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 is an alternative to `justification=unhyphenated`.

**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.polytonic]{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=** *<script-name>*

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

**language=** *<language-name>*

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

**alph=** *<counter-name>*

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

**Alph=** *<counter-name>*

Same for `\Alph`.

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

**onchar=** ids | fonts | 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 T<sub>E</sub>X sense (i. e., with catcode 11). Digits and punctuation are then considered part of current locale (as set by a selector). This option is useful when the main script is non-Latin and there is a secondary one whose script is Latin.

**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.

**NOTE** There is no general rule to set the font for a punctuation mark, because it is a semantic decision and not a typographical one. Consider the following sentence: “سہ، دو، یک” are Persian numbers”. In this case the punctuation font must be the English one, even if the commas are surrounded by non-Latin letters. Quotation marks, parenthesis, etc., are even more complex. Several criteria are possible, like the main language (the default in `babel`), the first letter in the paragraph, or the surrounding letters, among others, but even so manual switching can be still necessary.

**intraspace=** *<base> <shrink> <stretch>*

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=** *<penalty>*

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=** *<transform-list>*

See section 1.21.

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

**New 3.59** There are currently 4 options. Note they are language dependent, so that they will not be applied to other languages.

The first one (unhyphenated) activates a line breaking mode that allows spaces to be stretched to arbitrary amounts. Although for European standards the result may look odd, in some writing systems, like Malayalam and other Indic scripts, this has been the customary (although not always the desired) practice. Because of that, no locale sets currently this mode by default (Amharic is an exception). Unlike `\sloppy`, the `\hfuzz` and the `\vfuzz` are not changed, because this line breaking mode is not really ‘sloppy’ (in other words, overfull boxes are reported as usual).

The second and the third are 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.

```
\localnumeral {\style}{\number}
\localecounter {\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):

- `\localnumeral{\style}{\number}`, like `\localnumeral{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, kebona, kembata, konso, kunama, meen, oromo, saho, sidama, silti, tigre, wolaita, yemsa  
**Arabic** abjad, maghrebi.abjad  
**Armenian** lower.letter, upper.letter  
**Belarusian, 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. Çileyê Pêşîn 2019*, but with variant=iza fa it prints *31'ê Çileyê 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***name* 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 `\babel font` 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  $\text{\TeX}$  are entered as `-`, and (2) *optional* or *soft hyphens*, which are entered as `\-`. Strictly, a *soft hyphen* is not a hyphen, but just a breaking opportunity or, in  $\text{\TeX}$  terms, a “discretionary”; a *hard hyphen* is a hyphen with a breaking opportunity after it. A further type is a *non-breaking hyphen*, a hyphen without a breaking opportunity.

In  $\TeX$ , - and \- forbid further breaking opportunities in the word. This is the desired behavior very often, but not always, and therefore many languages provide shorthands for these cases. Unfortunately, this has not been done consistently: for example, " - in Dutch, Portuguese, Catalan or Danish is a hard hyphen, while in German, Spanish, Norwegian, Slovak or Russian is a soft hyphen. Furthermore, some of them even redefine \-, so that you cannot insert a soft hyphen without breaking opportunities in the rest of the word. Therefore, some macros are provided with a set of basic “hyphens” which can be used by themselves, to define a user shorthand, or even in language files.

- `\babelhyphen{soft}` and `\babelhyphen{hard}` are self explanatory.
- `\babelhyphen{repeat}` inserts a hard hyphen which is repeated at the beginning of the next line, as done in languages like Polish, Portuguese and Spanish.
- `\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  $\LaTeX$ : (1) the character used is that set for the current font, while in  $\LaTeX$  it is hardwired to - (a typical value); (2) the hyphen to be used in fonts with a negative `\hyphenchar` is -, like in  $\LaTeX$ , but it can be changed to another value by redefining `\babelnullhyphen`; (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 language \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 language \rangle, \langle language \rangle, ...] {\langle patterns \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:

```
\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
```

<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.

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	transliteration.dad	Applies the transliteration system devised by Yannis Haralambous for dad (simple and T <sub>E</sub> X-friendly). Not yet complete, but sufficient for most texts.
Croatian	digraphs.ligatures	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	hyphen.repeat	Explicit hyphens behave like <code>\babelhyphen{repeat}</code> .
Czech, Polish, Slovak	oneletter.nobreak	Converts a space after a non-syllabic preposition or conjunction into a non-breaking space.
Finnish	prehyphen.nobreak	Line breaks just after hyphens prepended to words are prevented, like in “pakastekaapit ja -arkut”.
Greek	diaeresis.hyphen	Removes the diaeresis above iota and upsilon if hyphenated just before. It works with the three variants.
Greek	transliteration.omega	Although the provided combinations are not the full set, this transform follows the syntax of Omega: = for the circumflex, v for digamma, and so on. For better compatibility with Levy’s system, ~ (as ‘string’) is an alternative to =. ' is tonos in Monotonic Greek, but oxia in Polytonic and Ancient Greek.
Greek	sigma.final	The transliteration system above does not convert the sigma at the end of a word (on purpose). This transforms does it. To prevent the conversion (an abbreviation, for example), write "s.
Hindi, Sanskrit	transliteration.hk	The Harvard-Kyoto system to romanize Devanagari.
Hindi, Sanskrit	punctuation.space	Inserts a space before the following four characters: !?;:.
Hungarian	digraphs.hyphen	Hyphenates the long digraphs <i>ccs, ddz, ggy, lly, nny, ssz, tty</i> and <i>zzs</i> as <i>cs-cs, dz-dz</i> , etc.
Indic scripts	danda.nobreak	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	digraphs.ligatures	Replaces the groups <i>ae, AE, oe, OE</i> with <i>æ, Æ, œ, Œ</i> .
Latin	letters.noj	Replaces <i>j, J</i> with <i>i, I</i> .
Latin	letters.uv	Replaces <i>v, U</i> with <i>u, V</i> .



Sanskrit	transliteration.iast	The IAST system to romanize Devanagari. <sup>16</sup>
Serbian	transliteration.gajica	(Note serbian with ini files refers to the Cyrillic script, which is here the target.) The standard system devised by Ljudevit Gaj.
Arabic, Persian	kashida.plain	Experimental. A very simple and basic transform for ‘plain’ Arabic fonts, which attempts to distribute the tatwil as evenly as possible (starting at the end of the line). See the news for version 3.59.

**\babelposthyphenation** [*<options>*]{*<hyphenrules-name>*}{*<lua-pattern>*}{*<replacement>*}

**New 3.37-3.39** With *luatex* it is possible to define non-standard hyphenation rules, like  $f-f \rightarrow ff-f$ , repeated hyphens, ranked ruled (or more precisely, ‘penalized’ hyphenation points), and so on. A few rules are currently provided (see above), but they can be defined as shown in the following example, where {1} is the first captured char (between ( ) in the pattern):

```
\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 ([ $\acute{u}$ ]), the replacement could be {1| $\acute{u}$ | $\acute{o}$ }, which maps  $\acute{t}$  to  $\acute{l}$ , and  $\acute{v}$  to  $\acute{o}$ , 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  $\acute{z}$  as zh and  $\acute{s}$  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 `\babelfont`. 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`  $\{ \langle text \rangle \}$

**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 `\TeX` and `\LaTeX` so that they are correctly typeset even with `LGR` or `X2` (the complete list is stored in `\BabelNonASCII`, which by default is `LGR`, `X2`, `OT2`, `OT3`, `OT6`, `LHE`, `LWN`, `LMA`, `LMC`, `LMS`, `LMU`, but you can modify it). So, in some sense it fixes the bug described in the previous paragraph.

If non-ASCII encodings are not loaded (or no encoding at all), it is no-op (also `\TeX` and `\LaTeX` are not redefined); otherwise, `\ensureascii` switches to the encoding at the beginning of the document if ASCII-savvy, or else the last ASCII-savvy encoding loaded. For example, if you load `LY1`, `LGR`, then it is set to `LY1`, but if you load `LY1`, `T2A` it is set to `T2A`.

<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.

The symbol encodings TS1, T3, and TS3 are not taken into account, since they are not used for “ordinary” text (they are stored in \BabelNonText, used in some special cases when no Latin encoding is explicitly set).

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

## 1.24 Selecting directions

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

**WARNING** The current code for **text** in **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 \textit{fuṣḥā l-‘aṣr} (MSA) and
\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`).`\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.

**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 T<sub>E</sub>X 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}
```

**`\babelsublr`** `{\lr-text}`

Digits in pdftex must be marked up explicitly (unlike luatex with `bidi=basic` or `bidi=basic-r` and, usually, xetex). This command is provided to set `{\lr-text}` in L mode if necessary. It's intended for what Unicode calls weak characters, because words are best set with the corresponding language. For this reason, there is no `rl` counterpart. Any `\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` $\{\langle section-name \rangle\}$

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

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

**New 3.17** Something like:

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

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

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

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

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

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

## 1.25 Language attributes

### `\languageattribute`

This is a user-level command, to be used in the preamble of a document (after `\usepackage[...]{babel}`), that declares which attributes are to be used for a given

language. It takes two arguments: the first is the name of the language; the second, a (list of) attribute(s) to be used. Attributes must be set in the preamble and only once – they cannot be turned on and off. The command checks whether the language is known in this document and whether the attribute(s) are known for this language.

Very often, using a *modifier* in a package option is better.

Several language definition files use their own methods to set options. For example, french uses `\frenchsetup`, magyar (1.5) uses `\magyarOptions`; modifiers provided by spanish have no attribute counterparts. Macros setting options are also used (eg, `\ProsodicMarksOn` in latin).

## 1.26 Hooks

**New 3.9a** A hook is a piece of code to be executed at certain events. Some hooks are predefined when `luatex` and `xetex` are used.

**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 `\usesshortands*` 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  $\TeX$  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>19</sup>  
**Romanian** romanian  
**Russian** russian  
**Scottish Gaelic** scottish  
**Spanish** spanish  
**Slovakian** slovak  
**Slovenian** slovene  
**Swedish** swedish  
**Serbian** serbian  
**Turkish** turkish  
**Ukrainian** ukrainian  
**Upper Sorbian** upporsorbian  
**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  $\text{\LaTeX}$ .

## 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).

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

**New 3.32** Here,  $\{\langle char-code \rangle\}$  is a number (with  $\text{\TeX}$  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).

<sup>19</sup>The two last name comes from the times when they had to be shortened to 8 characters

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` {<key-value-list>}

**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>20</sup> So, if you write a chunk of French text with \foreignlanguage, the

<sup>20</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.

apostrophes might not be taken into account. This is a limitation of  $\TeX$ , not of babel. Alternatively, you may use `\useshorthands` to activate ' and `\defineshortand`, 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  $\TeX$  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>21</sup> But that is the easy part, because they don't require modifying the  $\LaTeX$  internals.

Calendars (Arabic, Persian, Indic, etc.) are under study.

Also interesting are differences in the sentence structure or related to it. For example, in Basque the number precedes the name (including chapters), in Hungarian "from (1)" is "(1)-ből", but "from (3)" is "(3)-ből", in Spanish an item labelled "3." may be referred to as either "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.

<sup>21</sup>See for example POSIX, ISO 14652 and the Unicode Common Locale Data Repository (CLDR). Those systems, however, have limited application to  $\TeX$  because their aim is just to display information and not fine typesetting.

### 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`

$\TeX$  and most engines based on it (`pdf $\TeX$` , `xetex`,  $\epsilon$ - $\TeX$ , the main exception being `luatex`) require hyphenation patterns to be preloaded when a format is created (eg,  $\LaTeX$ ,  $\Xe\LaTeX$ , `pdf $\LaTeX$` ). `babel` provides a tool which has become standard in many distributions and based on a “configuration file” named `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>22</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>23</sup>

### 2.1 Format

In that file the person who maintains a  $\TeX$  environment has to record for which languages he has hyphenation patterns *and* in which files these are stored<sup>24</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  $\LaTeX$  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
=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>25</sup> For example:

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

<sup>22</sup>This feature was added to 3.9o, but it was buggy. Both 3.9o and 3.9p are deprecated.

<sup>23</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>24</sup>This is because different operating systems sometimes use *very* different file-naming conventions.

<sup>25</sup>This is not a new feature, but in former versions it didn't work correctly.

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`, `\captions<lang>`, `\date<lang>`, `\extras<lang>` and `\noextras<lang>` (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, `\date<lang>` but not `\captions<lang>` 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.
- 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  $\text{\LaTeX}$  (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>26</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.
- `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

<sup>26</sup>But not removed, for backward compatibility.

plain.tex version 3.x. Here “language” is used in the  $\TeX$  sense of set of hyphenation patterns.

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

**\<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  $\TeX$  might be in after the execution of `\extras<lang>`, a macro that brings  $\TeX$  into a predefined state is needed. It will be no surprise that the name of this macro is `\noextras<lang>`.

**\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  $\TeX$  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]
\LdfInit{<language>}{captions<language>}

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

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

\bb1@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:

<code>\AtEndOfPackage{%</code>	
<code>  \RequirePackage{dingbat}%</code>	Delay package
<code>  \savebox{\myeye}{\eye}}%</code>	And direct usage
<code>\newsavebox{\myeye}</code>	
<code>\newcommand\myanchor{\anchor}%</code>	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>27</sup>.

- `\babel@save` To save the current meaning of any control sequence, the macro `\babel@save` is provided. It takes one argument, `⟨csname⟩`, 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`. 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.

<sup>27</sup>This mechanism was introduced by Bernd Raichle.

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.

### 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 consists of 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`  $\langle\langle\textit{language-list}\rangle\rangle\langle\langle\textit{category}\rangle\rangle[\langle\langle\textit{selector}\rangle\rangle]$

The  $\langle\langle\textit{language-list}\rangle\rangle$  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 – an 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  $\langle category \rangle$  is either captions, date or extras. You must stick to these three categories, even if no error is raised when using other name.<sup>28</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\monthviiiname{August}
  \SetString\monthixname{September}
  \SetString\monthxname{Oktober}
  \SetString\monthxiname{November}
  \SetString\monthxiiname{Dezenber}
  \SetString\today{\number\day.-%
    \csname month\romannumeral\month name\endcsname\space
    \number\year}

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

\EndBabelCommands
```

When used in ldf files, previous values of  $\langle category \rangle \langle language \rangle$  are overridden, which means the old way to define strings still works and used by default (to be precise, is first set to undefined and then strings are added). However, when used in the preamble or in a package, new settings are added to the previous ones, if the language exists (in the babel sense, ie, if  $\langle date \rangle \langle language \rangle$  exists).

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

<sup>28</sup>In future releases further categories may be added.

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>29</sup>

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

**\AfterBabelCommands**  $\langle code \rangle$

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

**\SetString**  $\langle macro-name \rangle \langle string \rangle$

Adds  $\langle macro-name \rangle$  to the current category, and defines globally  $\langle lang-macro-name \rangle$  to  $\langle code \rangle$  (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**  $\langle macro-name \rangle \langle string-list \rangle$

A convenient way to define several ordered names at once. For example, to define \abmoniname, \abmoniiname, etc. (and similarly with abday):

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

#1 is replaced by the roman numeral.

**\SetCase**  $[\langle map-list \rangle] \langle toupper-code \rangle \langle tolower-code \rangle$

Sets globally code to be executed at \MakeUppercase and \MakeLowercase. The code would typically be things like \let\BB\bb and \uccode or \lccode (although for the reasons explained above, changes in lc/uc codes may not work). A  $\langle map-list \rangle$  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 L<sup>A</sup>T<sub>E</sub>X, we can set for Turkish:

```
\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
```

<sup>29</sup>This replaces in 3.9g a short-lived \UseStrings which has been removed because it did not work.

(Note the mapping for OT1 is not complete.)

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

**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`  $\{\langle uccode \rangle\}\{\langle lccode \rangle\}$  is similar to `\lccode` but it's ignored if the char has been set and saves the original `lccode` to restore it when switching the language (except with `hyphenmap=first`).
- `\BabelLowerMM`  $\{\langle uccode\text{-}from \rangle\}\{\langle uccode\text{-}to \rangle\}\{\langle step \rangle\}\{\langle lccode\text{-}from \rangle\}$  loops though the given uppercase codes, using the step, and assigns them the `lccode`, which is also increased (MM stands for *many-to-many*).
- `\BabelLowerMO`  $\{\langle uccode\text{-}from \rangle\}\{\langle uccode\text{-}to \rangle\}\{\langle step \rangle\}\{\langle lccode \rangle\}$  loops though the given uppercase codes, using the step, and assigns them the `lccode`, which is fixed (MO stands for *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`  $\{\langle selectors \rangle\}\{\langle true \rangle\}\{\langle false \rangle\}$

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

## 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  $\LaTeX$  package, which sets options and loads language styles.

**plain.def** defines some  $\LaTeX$  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 appropriate 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=3.83.2938>>
```

```
2 <<date=2022/12/01>>
```

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

We define some basic macros which just make the code cleaner. `\bbl@add` is now used internally instead of `\addto` because of the unpredictable behavior of the latter. Used in `babel.def` and in `babel.sty`, which means in  $\LaTeX$  is executed twice, but we need them when defining options and `babel.def` cannot be loaded 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@cl#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** 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>30</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{%
34   \begingroup
35   \let\<\<\noexpand
36   \let\<\bbl@exp@en
37   \let\[\bbl@exp@ue
38   \edef\bbl@exp@aux{\endgroup#1}%
39   \bbl@exp@aux}
40 \def\bbl@exp@en#1>{\expandafter\noexpand\csname#1\endcsname}%
41 \def\bbl@exp@ue#1]{%
42   \unexpanded\expandafter\expandafter\expandafter{\csname#1\endcsname}}%

```

**\bbl@trim** The following piece of code is stolen (with some changes) from `keyval`, by David Carlisle. It defines two macros: `\bbl@trim` and `\bbl@trim@def`. The first one strips the leading and trailing spaces from the second argument and then applies the first argument (a macro, `\toks@` and the like). The second one, as its name suggests, defines the first argument as the stripped second argument.

```

43 \def\bbl@tempa#1{%
44   \long\def\bbl@trim##1#2{%
45     \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
46   \def\bbl@trim@c{%
47     \ifx\bbl@trim@a\@sptoken
48       \expandafter\bbl@trim@b
49     \else
50       \expandafter\bbl@trim@b\expandafter#1%

```

<sup>30</sup>This code is based on code presented in TUGboat vol. 12, no2, June 1991 in “An expansion Power Lemma” by Sonja Maus.

```

51 \fi}%
52 \long\def\bbl@trim@b#1##1 \@nil{\bbl@trim@i##1}}
53 \bbl@tempa{ }
54 \long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
55 \long\def\bbl@trim@def#1{\bbl@trim{\def#1}}

```

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

```

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

```

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

```

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

```

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

```

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 checking 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 pdfTeX, 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}> ≡
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 L<sup>A</sup>TeX reserves for this purpose the count 19.

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

```

209 <{*Define core switching macros}> ≡
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 (L<sup>A</sup>TeX, `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\bb1@infowarn#1{%
241 \begingroup
242 \def\{\MessageBreak}%
243 \PackageNote{babel}{#1}%
244 \endgroup}
245 \def\bb1@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\bb1@info\@gobble
253 \let\bb1@infowarn\@gobble
254 \let\bb1@warning\@gobble}
255 {}
256 %
257 \def\AfterBabelLanguage#1{%
258 \global\expandafter\bb1@add\csname#1.ldf-h@@k\endcsname}%

```

If the format created a list of loaded languages (in \bb1@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\bb1@languages\undefined\else
260 \begingroup
261 \catcode\^^I=12
262 \@ifpackagewith{babel}{showlanguages}{%
263 \begingroup
264 \def\bb1@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
265 \wlog{<*languages>}%
266 \bb1@languages
267 \wlog{</languages>}%
268 \endgroup}{%
269 \endgroup
270 \def\bb1@elt#1#2#3#4{%
271 \ifnum#2=\z@
272 \gdef\bb1@nulllanguage{#1}%
273 \def\bb1@elt##1##2##3##4{}}%
274 \fi}%
275 \bb1@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 L<sup>A</sup>T<sub>E</sub>X 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 \bb1@trace{Defining option 'base'}
278 \@ifpackagewith{babel}{base}{%
279 \let\bb1@onlyswitch\@empty
280 \let\bb1@provide@locale\relax
281 \input babel.def
282 \let\bb1@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{1}%
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{1@#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}%
492     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
493     {}%
494   \ifx\bbl@bcp\relax
495     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
496   \fi
497   \else
498     \bbl@bcpcase#2\@empty\@empty\@#5\bbl@tempb
499     \bbl@bcpcase#3\@empty\@empty\@#5\bbl@tempc
500     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
501     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
502     {}%
503   \ifx\bbl@bcp\relax
504     \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
505     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
506     {}%
507   \fi
508   \ifx\bbl@bcp\relax
509     \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
510     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
511     {}%
512   \fi
513   \ifx\bbl@bcp\relax
514     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
515   \fi
516   \fi\fi}
517 \let\bbl@initoload\relax

```

```

518 \def\bbl@provide@locale{%
519   \ifx\babelprovide\undefined
520     \bbl@error{For a language to be defined on the fly 'base'\\%
521               is not enough, and the whole package must be\\%
522               loaded. Either delete the 'base' option or\\%
523               request the languages explicitly}%
524     {See the manual for further details.}%
525   \fi
526   \let\bbl@auxname\language % Still necessary. TODO
527   \bbl@ifunset{bbl@bcp@map@\language}{}}% Move uplevel??
528   {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
529   \ifbbl@bcp@allowed
530     \expandafter\ifx\csname date\language\endcsname\relax
531       \expandafter
532       \bbl@bcp@lookup\language-\@empty-\@empty-\@empty\@@
533       \ifx\bbl@bcp\relax\else % Returned by \bbl@bcp@lookup
534         \edef\language{\bbl@bcp@prefix\bbl@bcp}%
535         \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
536         \expandafter\ifx\csname date\language\endcsname\relax
537           \let\bbl@initoload\bbl@bcp
538           \bbl@exp{\bbl@babelprovide[\bbl@autoload@bcpoptions]{\language}}%
539           \let\bbl@initoload\relax
540         \fi
541         \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
542       \fi
543     \fi
544   \fi
545   \expandafter\ifx\csname date\language\endcsname\relax
546     \IfFileExists{babel-\language.tex}%
547     {\bbl@exp{\bbl@babelprovide[\bbl@autoload@options]{\language}}}%
548     {}%
549   \fi}

```

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

```

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\selectlanguageL`. 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=\@cclv\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 `luatex`, 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\@localname\@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\@localname{??}%
626 \else
627 \scantokens\expandafter{\expandafter
628 \def\expandafter\@localname\expandafter{\@language}}%
629 \fi
630 \fi
631 \else
632 \def\@localname{#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@bcppallowed
651 \bbl@bcppallowedfalse
652 \def\select@language#1{% from set@, babel@aux
653 \ifx\bbl@selectorname\empty
654 \def\bbl@selectorname{select}%
655 % set hmap
656 \fi
657 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\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@usedategroup
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\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\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{lnbrk}}%
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\rightthyphenmin
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 `otherlanguage` environment can be used as an alternative to using the `\selectlanguage` declarative command. When you are typesetting a document which mixes left-to-right and right-to-left typesetting you have to use this environment in order to let things work as you expect them to. The `\ignorespaces` command is necessary to hide the environment when it is entered in horizontal mode.

```

758 \long\def\otherlanguage#1{%
759   \def\bbl@selectorname{other}%
760   \ifnum\bbl@hymapsel=\@cclv\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 `otherlanguage` environment is meant to be used when a large part of text from a different language needs to be typeset, but without changing the translation of words such as ‘figure’. This environment makes use of `\foreign@language`.

```

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=\@cclv\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   \beginingroup
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   \beginingroup
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#1}%
812   \let\bbl@select@type\@ne
813   \expandafter\bbl@switch\expandafter{\language}

```

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 `\lccode's` has been set, too). `\bbl@hyphenation@` is set to relax until the very first `\babelhyphenation`, so do nothing with this value. If the exceptions for a language (by its number, not its name, so that `:ENC` is taken into account) has been set, then use `\hyphenation` with both global and language exceptions and empty the latter to mark they must not be set again.

```

821 \let\bbl@hyphlist\@empty

```



```

822 \let\bbl@hyphenation@\relax
823 \let\bbl@pttnlist\@empty
824 \let\bbl@patterns@\relax
825 \let\bbl@hmapsel=\@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, `\lccode`'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\languageshorthands\undefined\else
855       \languageshorthands{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 `\(lang)hyphenmins` 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\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,##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\bbl@select@type
1033 \bbl@cl{e}%
1034 \fi}%
1035 \begingroup
1036 \let\bbl@ens@include\@empty
1037 \let\bbl@ens@exclude\@empty
1038 \def\bbl@ens@fontenc{\relax}%
1039 \def\bbl@tempb##1{%
1040 \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1041 \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1042 \def\bbl@tempb##1=##2\@@{\@namedef{\bbl@ens@##1}{##2}}%
1043 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
1044 \def\bbl@tempc{\bbl@ensure}%
1045 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1046 \expandafter{\bbl@ens@include}}%
1047 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1048 \expandafter{\bbl@ens@exclude}}%
1049 \toks@\expandafter{\bbl@tempc}%
1050 \bbl@exp{%

```

```

1051 \endgroup
1052 \def\<bbl@e#2>\the\toks@\bbl@ens@fontenc}}
1053 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1054 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1055 \ifx##1\undefined % 3.32 - Don't assume the macro exists
1056 \edef##1{\noexpand\bbl@nocaption
1057 {\bbl@stripslash##1}\language\language\bbl@stripslash##1}}%
1058 \fi
1059 \ifx##1\empty\else
1060 \in@{##1}{#2}%
1061 \ifin\else
1062 \bbl@ifunset{\bbl@ensure@\language}%
1063 {\bbl@exp{%
1064 \\\DeclareRobustCommand\<bbl@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 \bbl@csarg\noexpand{ensure@\language}%
1074 {\the\toks@}}%
1075 \fi
1076 \expandafter\bbl@tempb
1077 \fi}%
1078 \expandafter\bbl@tempb\bbl@captionslist\today\empty
1079 \def\bbl@tempa##1{% elt for include list
1080 \ifx##1\empty\else
1081 \bbl@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
1082 \ifin\else
1083 \bbl@tempb##1\empty
1084 \fi
1085 \expandafter\bbl@tempa
1086 \fi}%
1087 \bbl@tempa#1\empty}
1088 \def\bbl@captionslist{%
1089 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1090 \contentsname\listfigurename\listtablename\indexname\figurename
1091 \tablename\partname\enclname\ccname\headtoname\pagename\seename
1092 \alsoaname\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 at-sign. We make sure that it is a ‘letter’ during the processing of the file. We also save its name as the last called option, even if not loaded.

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

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

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

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

```

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 `LTEX`.

```

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\bbbl@main@language % TODO. Set locale name
1142 \bbbl@id@assign
1143 \bbbl@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\bbbl@beforestart{%
1145   \def\nolanerr##1{%
1146     \bbbl@warning{Undefined language '##1' in aux.\\Reported}}%
1147   \bbbl@usehooks{beforestart}{}%
1148   \global\let\bbbl@beforestart\relax}
1149 \AtBeginDocument{%
1150   {\nameuse{bbbl@beforestart}}% Group!
1151   \iffiles
1152     \providecommand\babel@aux[2]{}%
1153     \immediate\write\@mainaux{\string\babel@aux[2]}%
1154     \string\providecommand\string\babel@aux[2]}%
1155     \immediate\write\@mainaux{\string\nameuse{bbbl@beforestart}}%
1156   \fi
1157   \expandafter\selectlanguage\expandafter{\bbbl@main@language}%
1158   \ifbbbl@single % must go after the line above.
1159     \renewcommand\selectlanguage[1]{}%
1160     \renewcommand\foreignlanguage[2]{#2}%
1161     \global\let\babel@aux\@gobbles % Also as flag
1162   \fi
1163   \ifcase\bbbl@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\bbbl@select@type
1166     \bbbl@ifsamestring\language{#1}{\select@language{#1}}%
1167   \else
1168     \select@language{#1}%
1169   \fi}

```

## 7.5 Shorthands

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

```

1170 \bbbl@trace{Shorthands}
1171 \def\bbbl@add@special#1{% 1:a macro like \", \?, etc.
1172   \bbbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1173   \bbbl@ifunset{@sanitize}{\bbbl@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         \bbbl@add\nfss@catcodes{\makeother#1}%
1181       \else
1182         \endgroup
1183       \fi
1184     \fi}

```

`\bbbl@remove@special` The companion of the former macro is `\bbbl@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#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
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@m@s` as well. Also, make sure that a single `'` in math mode ‘does the right thing’. (2) If we are using the caret (`^`) as a shorthand character special care should be taken to make sure math still works. Therefore an extra level of expansion is introduced with a check for math mode on the upper level.

```
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\bbl@firstcs
1368     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1369     {\def\bbl@tempa{#4}%
1370      \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1371      \else
1372        \bbl@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 \ifmode
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@usesesh@
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\bb1@tempa{\zap@space#1 \@empty}%
1415   \bb1@for\bb1@tempb\bb1@tempa{%
1416     \if*\expandafter\@car\bb1@tempb\@nil
1417       \edef\bb1@tempb{user@\expandafter\@gobble\bb1@tempb}%
1418       \@expandtwoargs
1419       \bb1@set@user@generic{\expandafter\string\@car#2\@nil}\bb1@tempb
1420     \fi
1421     \declare@shorthand{\bb1@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 shorhands 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  $\TeX$ ) 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@attrs`. When that control sequence is not yet defined this attribute is certainly not selected before.

```

1539     \ifx\bbl@known@attrs\undefined
1540       \in@false
1541     \else
1542       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
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  $\TeX$ -code.

```

1549     \bbl@exp{%
1550       \\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
1551     \edef\bbl@tempa{\bbl@tempc-##1}%
1552     \expandafter\bbl@ifknown@attrib\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 TeX code based on whether a certain attribute was set. This command should appear inside the argument to `\AtBeginDocument` because the attributes are set in the document preamble, *after* babel is loaded. The first argument is the language, the second argument the attribute being checked, and the third and fourth arguments are the true and false clauses.

```
1568 \def\bbl@ifattributeset#1#2#3#4{%
1569   \ifx\bbl@known@attrs\undefined
1570     \in@false
1571   \else
1572     \bbl@xin@{,#1-#2,}{,\bbl@known@attrs,}%
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 TeX-code to be executed when the attribute is known and the TeX-code to be executed otherwise. We first assume the attribute is unknown. Then we loop over the list of known attributes, trying to find a match.

```
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 TeX'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⟨csmame⟩` saves the current meaning of the control sequence `⟨csmame⟩` to `\originalTeX`<sup>31</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\scode`\.=\@m`  
1613 `\let\bbl@nonfrenchspacing\relax`  
1614 `\else`  
1615 `\frenchspacing`  
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{\scode`##1=\the\scode`##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 `\elseif n\bbl@tempa % non french`  
1633 `\def\bbl@elt##1##2##3{%`

<sup>31</sup>`\originalTeX` has to be expandable, i.e. you shouldn't let it to `\relax`.

```

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\noexpand\csname ##1\endcsname{%
1653 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1654 \noexpand\protect
1655 \expandafter\noexpand\csname text##1\endcsname{%
1656 \expandafter\noexpand\foreignlanguage{##2}}}%
1657 \bbl@tempc}%
1658 \bbl@for\bbl@tempa\bbl@tempa{%
1659 \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>32</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}

```

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{}{}{}}

```

<sup>32</sup>`TEX` begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

`\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@tglobal#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 `\lang\bbl@uclc` because we do not know how many expansions are necessary (depends on whether strings are encoded). The last part is tricky – when uppercasing, we have:

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

and starts over (and similarly when lowercasing).

```
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, for testing purposes:
1743 \def\BabelRestoreCase{%
1744   \DeclareRobustCommand{\MakeUppercase}[1]{%
1745     \def\reserved@a####1####2{\let####1####2\reserved@a}%
1746     \def\i{I}\def\j{J}%
1747     \expandafter\reserved@a\@uclclist\reserved@b{\reserved@b\@gobble}%
1748     \let\UTF@two@octets@noexpand\empty
1749     \let\UTF@three@octets@noexpand\empty
1750     \let\UTF@four@octets@noexpand\empty
1751     \protected@edef\reserved@a{\uppercase{##1}}%
1752     \reserved@a
1753   }%
1754   \DeclareRobustCommand{\MakeLowercase}[1]{%
1755     \def\reserved@a####1####2{\let####2####1\reserved@a}%
1756     \expandafter\reserved@a\@uclclist\reserved@b{\reserved@b\@gobble}%
1757     \let\UTF@two@octets@noexpand\empty
1758     \let\UTF@three@octets@noexpand\empty
1759     \let\UTF@four@octets@noexpand\empty
1760     \protected@edef\reserved@a{\lowercase{##1}}%
1761     \reserved@a}}
1762 <<(*More package options)>> ≡
1763 \DeclareOption{nocase}{}
1764 <</More package options>>
```

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

```

1765 <<{*More package options}>> ≡
1766 \let\bbl@opt@strings\@nnil % accept strings=value
1767 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1768 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1769 \def\BabelStringsDefault{generic}
1770 <</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.

```

1771 \@onlypreamble\StartBabelCommands
1772 \def\StartBabelCommands{%
1773   \begingroup
1774   \@tempcnta="7F
1775   \def\bbl@tempa{%
1776     \ifnum\@tempcnta>"FF\else
1777       \catcode\@tempcnta=11
1778       \advance\@tempcnta\@ne
1779       \expandafter\bbl@tempa
1780     \fi}%
1781   \bbl@tempa
1782   <<Macros local to BabelCommands>>
1783   \def\bbl@provstring##1##2{%
1784     \providecommand##1{##2}%
1785     \bbl@toglobal##1}%
1786   \global\let\bbl@scafter\@empty
1787   \let\StartBabelCommands\bbl@startcmds
1788   \ifx\BabelLanguages\relax
1789     \let\BabelLanguages\CurrentOption
1790   \fi
1791   \begingroup
1792   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1793   \StartBabelCommands}
1794 \def\bbl@startcmds{%
1795   \ifx\bbl@screset\@nnil\else
1796     \bbl@usehooks{stopcommands}{}}%
1797   \fi
1798   \endgroup
1799   \begingroup
1800   \@ifstar
1801     {\ifx\bbl@opt@strings\@nnil
1802       \let\bbl@opt@strings\BabelStringsDefault
1803       \fi
1804       \bbl@startcmds@i}%
1805     \bbl@startcmds@i}
1806 \def\bbl@startcmds@i#1#2{%
1807   \edef\bbl@L{\zap@space#1 \@empty}%
1808   \edef\bbl@G{\zap@space#2 \@empty}%
1809   \bbl@startcmds@ii}
1810 \let\bbl@startcmds\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.

```

1811 \newcommand\bbl@startcmds@ii[1][\@empty]{%

```

```

1812 \let\SetString\@gobbletwo
1813 \let\bbl@stringdef\@gobbletwo
1814 \let\AfterBabelCommands\@gobble
1815 \ifx\@empty#1%
1816   \def\bbl@sc@label{generic}%
1817   \def\bbl@encstring##1##2{%
1818     \ProvideTextCommandDefault##1{##2}%
1819     \bbl@tglobal##1%
1820     \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%
1821   \let\bbl@sctest\in@true
1822 \else
1823   \let\bbl@sc@charset\space % <- zapped below
1824   \let\bbl@sc@fontenc\space % <- " "
1825   \def\bbl@tempa##1=##2\@nil{%
1826     \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1827   \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1828   \def\bbl@tempa##1 ##2{% space -> comma
1829     ##1%
1830     \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1831   \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1832   \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1833   \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1834   \def\bbl@encstring##1##2{%
1835     \bbl@foreach\bbl@sc@fontenc{%
1836       \bbl@ifunset{T@####1}%
1837       }%
1838     {\ProvideTextCommand##1{####1}{##2}%
1839     \bbl@tglobal##1%
1840     \expandafter
1841     \bbl@tglobal\csname####1\string##1\endcsname}}}%
1842   \def\bbl@sctest{%
1843     \bbl@xin@{\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1844   \fi
1845   \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
1846   \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
1847     \let\AfterBabelCommands\bbl@aftercmds
1848     \let\SetString\bbl@setstring
1849     \let\bbl@stringdef\bbl@encstring
1850   \else % ie, strings=value
1851     \bbl@sctest
1852   \ifin@
1853     \let\AfterBabelCommands\bbl@aftercmds
1854     \let\SetString\bbl@setstring
1855     \let\bbl@stringdef\bbl@provstring
1856   \fi\fi\fi
1857   \bbl@scswitch
1858   \ifx\bbl@G\@empty
1859     \def\SetString##1##2{%
1860       \bbl@error{Missing group for string \string##1}%
1861       {You must assign strings to some category, typically\\%
1862       captions or extras, but you set none}}%
1863   \fi
1864   \ifx\@empty#1%
1865     \bbl@usehooks{defaultcommands}{}%
1866   \else
1867     \@expandtwoargs
1868     \bbl@usehooks{encodedcommands}{\bbl@sc@charset}\bbl@sc@fontenc}%
1869   \fi}

```

There are two versions of `\bbl@scswitch`. The first version is used when `ldfs` are read, and it makes sure `\group\language` is reset, but only once (`\bbl@screset` is used to keep track of this). The second version is used in the preamble and packages loaded after `babel` and does nothing. The macro `\bbl@forlang` loops `\bbl@L` but its body is executed only if the value is in `\BabelLanguages` (inside `babel`) or `\date\language` is defined (after `babel` has been loaded). There



are also two version of \bbl@forlang. The first one skips the current iteration if the language is not in \BabelLanguages (used in ldfs), and the second one skips undefined languages (after babel has been loaded) .

```

1870 \def\bbl@forlang#1#2{%
1871   \bbl@for#1\bbl@L{%
1872     \bbl@xin@{,#1,},{,\BabelLanguages,}%
1873     \ifin#2\relax\fi}}
1874 \def\bbl@scswitch{%
1875   \bbl@forlang\bbl@tempa{%
1876     \ifx\bbl@G@empty\else
1877       \ifx\SetString@gobbletwo\else
1878         \edef\bbl@GL{\bbl@G\bbl@tempa}%
1879         \bbl@xin@{,\bbl@GL,},{,\bbl@screset,}%
1880         \ifin@else
1881           \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1882           \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1883         \fi
1884       \fi
1885     \fi}}
1886 \AtEndOfPackage{%
1887   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
1888   \let\bbl@scswitch\relax}
1889 \@onlypreamble\EndBabelCommands
1890 \def\EndBabelCommands{%
1891   \bbl@usehooks{stopcommands}{}}%
1892   \endgroup
1893   \endgroup
1894   \bbl@scafter}
1895 \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.

```

1896 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1897   \bbl@forlang\bbl@tempa{%
1898     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1899     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1900     {\bbl@exp{%
1901       \global\bbbl@add\<\bbl@G\bbl@tempa>\bbbl@scset\#1\<\bbl@LC>}}}%
1902     {}}%
1903   \def\BabelString{#2}%
1904   \bbl@usehooks{stringprocess}{}}%
1905   \expandafter\bbl@stringdef
1906   \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.

```

1907 \ifx\bbl@opt@strings\relax
1908   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1909   \bbl@patchuclc
1910   \let\bbl@encoded\relax
1911   \def\bbl@encoded@uclc#1{%
1912     \@inmathwarn#1%
1913     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1914       \expandafter\ifx\csname ?\string#1\endcsname\relax
1915         \TextSymbolUnavailable#1%
1916       \else
1917         \csname ?\string#1\endcsname

```

```

1918     \fi
1919   \else
1920     \csname\cf@encoding\string#1\endcsname
1921   \fi}
1922 \else
1923   \def\bbl@scset#1#2{\def#1{#2}}
1924 \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.

```

1925 <<(*Macros local to BabelCommands)>> ≡
1926 \def\SetStringLoop##1##2{%
1927   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1928   \count@\z@
1929   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1930     \advance\count@ \@ne
1931     \toks@\expandafter{\bbl@tempa}%
1932     \bbl@exp{%
1933       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1934       \count@=\the\count@\relax}}}%
1935 <</Macros local to BabelCommands>>

```

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

```

1936 \def\bbl@aftercmds#1{%
1937   \toks@\expandafter{\bbl@scafter#1}%
1938   \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.

```

1939 <<(*Macros local to BabelCommands)>> ≡
1940 \newcommand\SetCase[3][{%
1941   \bbl@patchuclc
1942   \bbl@forlang\bbl@tempa{%
1943     \bbl@carg\bbl@encstring{\bbl@tempa @bbl@uclc}{\bbl@tempa##1}%
1944     \bbl@carg\bbl@encstring{\bbl@tempa @bbl@uc}{##2}%
1945     \bbl@carg\bbl@encstring{\bbl@tempa @bbl@lc}{##3}}}%
1946 <</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.

```

1947 <<(*Macros local to BabelCommands)>> ≡
1948 \newcommand\SetHyphenMap[1]{%
1949   \bbl@forlang\bbl@tempa{%
1950     \expandafter\bbl@stringdef
1951     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1952 <</Macros local to BabelCommands>>

```

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

```

1953 \newcommand\BabelLower[2]{% one to one.
1954   \ifnum\lccode#1=#2\else
1955     \babel@savevariable{\lccode#1}%
1956     \lccode#1=#2\relax
1957   \fi}
1958 \newcommand\BabelLowerMM[4]{% many-to-many
1959   \@tempcnta=#1\relax
1960   \@tempcntb=#4\relax
1961   \def\bbl@tempa{%
1962     \ifnum\@tempcnta>#2\else
1963       \expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%

```

```

1964      \advance\@tempcnta#3\relax
1965      \advance\@tempcntb#3\relax
1966      \expandafter\bb1@tempa
1967      \fi}%
1968      \bb1@tempa}
1969 \newcommand\BabelLowerMO[4]{% many-to-one
1970   \@tempcnta=#1\relax
1971   \def\bb1@tempa{%
1972     \ifnum\@tempcnta>#2\else
1973       \expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1974       \advance\@tempcnta#3
1975       \expandafter\bb1@tempa
1976     \fi}%
1977   \bb1@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1978 <<{*More package options}> \equiv
1979 \DeclareOption{hyphenmap=off}{\chardef\bb1@opt@hyphenmap\z@}
1980 \DeclareOption{hyphenmap=first}{\chardef\bb1@opt@hyphenmap\@ne}
1981 \DeclareOption{hyphenmap=select}{\chardef\bb1@opt@hyphenmap\tw@}
1982 \DeclareOption{hyphenmap=other}{\chardef\bb1@opt@hyphenmap\thr@@}
1983 \DeclareOption{hyphenmap=other*}{\chardef\bb1@opt@hyphenmap4\relax}
1984 <</More package options>>

```

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

```

1985 \AtEndOfPackage{%
1986   \ifx\bb1@opt@hyphenmap\undefined
1987     \bb1@xin@{,}{\bb1@language@opts}%
1988     \chardef\bb1@opt@hyphenmap\ifin@4\else\@ne\fi
1989   \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.

```

1990 \newcommand\setlocalecaption{% TODO. Catch typos.
1991   \@ifstar\bb1@setcaption@s\bb1@setcaption@x}
1992 \def\bb1@setcaption@x#1#2#3{% language caption-name string
1993   \bb1@trim@def\bb1@tempa{#2}%
1994   \bb1@xin@{.template}{\bb1@tempa}%
1995   \ifin@
1996     \bb1@ini@captions@template{#3}{#1}%
1997   \else
1998     \edef\bb1@tempd{%
1999       \expandafter\expandafter\expandafter
2000       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2001     \bb1@xin@
2002       {\expandafter\string\csname #2name\endcsname}%
2003       {\bb1@tempd}%
2004     \ifin@ % Renew caption
2005       \bb1@xin@{\string\bb1@scset}{\bb1@tempd}%
2006       \ifin@
2007         \bb1@exp{%
2008           \\bb1@ifsamestring{\bb1@tempa}{\language}%
2009           {\bb1@scset\<#2name>\<#1#2name>}%
2010           {}}%
2011         \else % Old way converts to new way
2012           \bb1@ifunset{#1#2name}%
2013             {\bb1@exp{%
2014               \\bb1@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2015               \\bb1@ifsamestring{\bb1@tempa}{\language}%
2016               {\def\<#2name>{\<#1#2name>}}%
2017               {}}}%
2018             {%
2019           \fi

```

```

2020 \else
2021 \bbl@xin@\string\bbl@scset{\bbl@tempd}% New
2022 \ifin@ % New way
2023 \bbl@exp{%
2024 \\\bbl@add\<captions#1>{\\\bbl@scset\<#2name>\<#1#2name>}%
2025 \\\bbl@ifsamestring{\bbl@tempa}{\language}%
2026 {\\\bbl@scset\<#2name>\<#1#2name>}%
2027 }%
2028 \else % Old way, but defined in the new way
2029 \bbl@exp{%
2030 \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2031 \\\bbl@ifsamestring{\bbl@tempa}{\language}%
2032 {\def\<#2name>{\<#1#2name>}}%
2033 }%
2034 \fi
2035 \fi
2036 \@namedef{#1#2name}{#3}%
2037 \toks@\expandafter{\bbl@captionslist}%
2038 \bbl@exp{\in@\<#2name>}{\the\toks@}%
2039 \ifin\else
2040 \bbl@exp{\\\bbl@add\\bbl@captionslist{\<#2name>}}%
2041 \bbl@tglobal\bbl@captionslist
2042 \fi
2043 \fi}
2044 % \def\bbl@setcaption#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.

```

2045 \bbl@trace{Macros related to glyphs}
2046 \def\set@low@box#1{\setbox\tw@hbox{,}\setbox\z@hbox{#1}%
2047 \dimen\z@ht\z@ \advance\dimen\z@ -\ht\tw@%
2048 \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.

```

2049 \def\save@sf@q#1{\leavevmode
2050 \begingroup
2051 \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2052 \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.

```

2053 \ProvideTextCommand{\quotedblbase}{OT1}{%
2054 \save@sf@q{\set@low@box{\textquotedblright\}}%
2055 \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.

```

2056 \ProvideTextCommandDefault{\quotedblbase}{%
2057 \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2058 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2059 \save@sf@q{\set@low@box{\textquoteright\}}%
2060 \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.

```
2061 \ProvideTextCommandDefault{\quotesinglbase}{%
2062   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2063 \ProvideTextCommand{\guillemetleft}{OT1}{%
2064   \ifmmode
2065     \ll
2066   \else
2067     \save@sf@q{\nobreak
2068       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2069   \fi}
2070 \ProvideTextCommand{\guillemetright}{OT1}{%
2071   \ifmmode
2072     \gg
2073   \else
2074     \save@sf@q{\nobreak
2075       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2076   \fi}
2077 \ProvideTextCommand{\guillemotleft}{OT1}{%
2078   \ifmmode
2079     \ll
2080   \else
2081     \save@sf@q{\nobreak
2082       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2083   \fi}
2084 \ProvideTextCommand{\guillemotright}{OT1}{%
2085   \ifmmode
2086     \gg
2087   \else
2088     \save@sf@q{\nobreak
2089       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2090   \fi}
```

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

```
2091 \ProvideTextCommandDefault{\guillemetleft}{%
2092   \UseTextSymbol{OT1}{\guillemetleft}}
2093 \ProvideTextCommandDefault{\guillemetright}{%
2094   \UseTextSymbol{OT1}{\guillemetright}}
2095 \ProvideTextCommandDefault{\guillemotleft}{%
2096   \UseTextSymbol{OT1}{\guillemotleft}}
2097 \ProvideTextCommandDefault{\guillemotright}{%
2098   \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
2099 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2100   \ifmmode
2101     <%
2102   \else
2103     \save@sf@q{\nobreak
2104       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2105   \fi}
2106 \ProvideTextCommand{\guilsinglright}{OT1}{%
2107   \ifmmode
2108     >%
2109   \else
2110     \save@sf@q{\nobreak
2111       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2112   \fi}
```

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

```
2113 \ProvideTextCommandDefault{\guilsinglleft}{%
```

```

2114 \UseTextSymbol{OT1}{\guilsinglleft}}
2115 \ProvideTextCommandDefault{\guilsinglright}{%
2116 \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.

```

2117 \DeclareTextCommand{\ij}{OT1}{%
2118 i\kern-0.02em\bbl@allowhyphens j}
2119 \DeclareTextCommand{\IJ}{OT1}{%
2120 I\kern-0.02em\bbl@allowhyphens J}
2121 \DeclareTextCommand{\ij}{T1}{\char188}
2122 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2123 \ProvideTextCommandDefault{\ij}{%
2124 \UseTextSymbol{OT1}{\ij}}
2125 \ProvideTextCommandDefault{\IJ}{%
2126 \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).

```

2127 \def\crrtic@{\hrule height0.1ex width0.3em}
2128 \def\crttic@{\hrule height0.1ex width0.33em}
2129 \def\ddj@{%
2130 \setbox0\hbox{d}\dimen@=\ht0
2131 \advance\dimen@1ex
2132 \dimen@.45\dimen@
2133 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2134 \advance\dimen@ii.5ex
2135 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2136 \def\DDJ@{%
2137 \setbox0\hbox{D}\dimen@=.55\ht0
2138 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2139 \advance\dimen@ii.15ex % correction for the dash position
2140 \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2141 \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2142 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2143 %
2144 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2145 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2146 \ProvideTextCommandDefault{\dj}{%
2147 \UseTextSymbol{OT1}{\dj}}
2148 \ProvideTextCommandDefault{\DJ}{%
2149 \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.

```

2150 \DeclareTextCommand{\SS}{OT1}{SS}
2151 \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
2152 \ProvideTextCommandDefault{\glq}{%
2153   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

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

2154 \ProvideTextCommand{\grq}{T1}{%
2155   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2156 \ProvideTextCommand{\grq}{TU}{%
2157   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2158 \ProvideTextCommand{\grq}{OT1}{%
2159   \save@sf@q{\kern-.0125em
2160     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2161     \kern.07em\relax}}
2162 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

`\glqq` The ‘german’ double quotes.

```
\grqq
2163 \ProvideTextCommandDefault{\glqq}{%
2164   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

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

2165 \ProvideTextCommand{\grqq}{T1}{%
2166   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2167 \ProvideTextCommand{\grqq}{TU}{%
2168   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2169 \ProvideTextCommand{\grqq}{OT1}{%
2170   \save@sf@q{\kern-.07em
2171     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2172     \kern.07em\relax}}
2173 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq` The ‘french’ single guillemets.

```
\frq
2174 \ProvideTextCommandDefault{\flq}{%
2175   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2176 \ProvideTextCommandDefault{\frq}{%
2177   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq
2178 \ProvideTextCommandDefault{\flqq}{%
2179   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2180 \ProvideTextCommandDefault{\frqq}{%
2181   \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).

```
2182 \def\umlauthigh{%
2183   \def\bbl@umlauta##1{\leavevmode\bgroup%
2184     \accent\csname\fontencoding dqpos\endcsname
2185     ##1\bbl@allowhyphens\egroup}%
2186   \let\bbl@umlaute\bbl@umlauta}
2187 \def\umlautlow{%
2188   \def\bbl@umlauta{\protect\lower@umlaut}}
2189 \def\umlautelow{%
2190   \def\bbl@umlaute{\protect\lower@umlaut}}
2191 \umlauthigh
```

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

```
2192 \expandafter\ifx\csname U@D\endcsname\relax
2193   \csname newdimen\endcsname\U@D
2194 \fi
```

The following code fools  $\text{T}_\text{E}\text{X}$ '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.

```
2195 \def\lower@umlaut#1{%
2196   \leavevmode\bgroup
2197   \U@D 1ex%
2198   {\setbox\z@\hbox{%
2199     \char\csname\fontencoding dqpos\endcsname}%
2200     \dimen@ -.45ex\advance\dimen@\ht\z@
2201     \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2202   \accent\csname\fontencoding dqpos\endcsname
2203   \fontdimen5\font\U@D #1%
2204   \egroup}
```

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

```
2205 \AtBeginDocument{%
2206   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2207   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2208   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2209   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{i}}%
2210   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
2211   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
2212   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
2213   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2214   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2215   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
2216   \DeclareTextCompositeCommand{\}{OT1}{U}{\bbl@umlauta{U}}}
```

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

```
2217 \ifx\l@english\undefined
2218   \chardef\l@english\z@
2219 \fi
2220 % The following is used to cancel rules in ini files (see Amharic).
2221 \ifx\l@unhyphenated\undefined
2222   \newlanguage\l@unhyphenated
2223 \fi
```

## 7.13 Layout

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

```
2224 \bbl@trace{Bidi layout}
2225 \providecommand\IfBabelLayout[3]{#3}%
2226 \newcommand\BabelPatchSection[1]{%
2227   \@ifundefined{#1}{%
2228     \bbl@exp{\let\bbl@ss#1>\<#1>}%
```



```

2229 \namedef{#1}{%
2230 \ifstar{\bbl@presec@s{#1}}%
2231 {\@dblarg{\bbl@presec@x{#1}}}}%
2232 \def\bbl@presec@x#1[#2]#3{%
2233 \bbl@exp{%
2234 \select@language@x{\bbl@main@language}%
2235 \bbl@cs{sspre@#1}%
2236 \bbl@cs{ss@#1}%
2237 [\foreignlanguage{\language}{\unexpanded{#2}}}%
2238 {\foreignlanguage{\language}{\unexpanded{#3}}}%
2239 \select@language@x{\language}}%
2240 \def\bbl@presec@s#1#2{%
2241 \bbl@exp{%
2242 \select@language@x{\bbl@main@language}%
2243 \bbl@cs{sspre@#1}%
2244 \bbl@cs{ss@#1}*%
2245 {\foreignlanguage{\language}{\unexpanded{#2}}}%
2246 \select@language@x{\language}}%
2247 \IfBabelLayout{sectioning}%
2248 {\BabelPatchSection{part}%
2249 \BabelPatchSection{chapter}%
2250 \BabelPatchSection{section}%
2251 \BabelPatchSection{subsection}%
2252 \BabelPatchSection{subsubsection}%
2253 \BabelPatchSection{paragraph}%
2254 \BabelPatchSection{subparagraph}%
2255 \def\babel@toc#1{%
2256 \select@language@x{\bbl@main@language}}}%
2257 \IfBabelLayout{captions}%
2258 {\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.

```

2259 \bbl@trace{Input engine specific macros}
2260 \ifcase\bbl@engine
2261 \input txtbabel.def
2262 \or
2263 \input luababel.def
2264 \or
2265 \input xebabel.def
2266 \fi
2267 \providecommand\babelfont{%
2268 \bbl@error
2269 {This macro is available only in LuaLaTeX and XeLaTeX.}%
2270 {Consider switching to these engines.}}
2271 \providecommand\babelprehyphenation{%
2272 \bbl@error
2273 {This macro is available only in LuaLaTeX.}%
2274 {Consider switching to that engine.}}
2275 \ifx\babelposthyphenation\@undefined
2276 \let\babelposthyphenation\babelprehyphenation
2277 \let\babelpatterns\babelprehyphenation
2278 \let\babelcharproperty\babelprehyphenation
2279 \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.

```

2280 \bbl@trace{Creating languages and reading ini files}
2281 \let\bbl@extend@ini\@gobble
2282 \newcommand\babelprovide[2][]{%
2283   \let\bbl@savelangname\language
2284   \edef\bbl@savelocaleid{\the\localeid}%
2285   % Set name and locale id
2286   \edef\language{#2}%
2287   \bbl@id@assign
2288   % Initialize keys
2289   \bbl@vforeach{captions,date,import,main,script,language,%
2290     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2291     mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2292     Alph,labels,labels*,calendar,date}%
2293     {\bbl@csarg\let{KVP@##1}\@nnil}%
2294   \global\let\bbl@release@transforms\@empty
2295   \let\bbl@calendars\@empty
2296   \global\let\bbl@inidata\@empty
2297   \global\let\bbl@extend@ini\@gobble
2298   \gdef\bbl@key@list{;}%
2299   \bbl@forkv{#1}{%
2300     \in@{/}{##1}%
2301     \ifin@
2302       \global\let\bbl@extend@ini\bbl@extend@ini@aux
2303       \bbl@renewinikey##1\@{##2}%
2304     \else
2305       \bbl@csarg\ifx{KVP@##1}\@nnil\else
2306         \bbl@error
2307           {Unknown key '##1' in \string\babelprovide}%
2308           {See the manual for valid keys}%
2309       \fi
2310       \bbl@csarg\def{KVP@##1}{##2}%
2311     \fi}%
2312   \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2313   \bbl@ifunset{date#2}\z@{\bbl@ifunset{\bbl@llevel#2}\@ne\tw@}%
2314   % == init ==
2315   \ifx\bbl@screset\@undefined
2316     \bbl@ldfinit
2317   \fi
2318   % == date (as option) ==
2319   % \ifx\bbl@KVP@date\@nnil\else
2320   % \fi
2321   % ==
2322   \let\bbl@lbkflag\relax % \@empty = do setup linebreak
2323   \ifcase\bbl@howloaded
2324     \let\bbl@lbkflag\@empty % new
2325   \else
2326     \ifx\bbl@KVP@hyphenrules\@nnil\else
2327       \let\bbl@lbkflag\@empty
2328     \fi
2329     \ifx\bbl@KVP@import\@nnil\else
2330       \let\bbl@lbkflag\@empty
2331     \fi
2332   \fi
2333   % == import, captions ==
2334   \ifx\bbl@KVP@import\@nnil\else
2335     \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
2336     {\ifx\bbl@initload\relax
2337       \begingroup
2338         \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2339         \bbl@input@texini{##2}%
2340       \endgroup
2341     \else
2342       \xdef\bbl@KVP@import{\bbl@initload}%

```

```

2343     \fi}%
2344     }%
2345     \let\bbl@KVP@date\@empty
2346 \fi
2347 \ifx\bbl@KVP@captions\@nnil
2348     \let\bbl@KVP@captions\bbl@KVP@import
2349 \fi
2350 % ==
2351 \ifx\bbl@KVP@transforms\@nnil\else
2352     \bbl@replace\bbl@KVP@transforms{ }{,}%
2353 \fi
2354 % == Load ini ==
2355 \ifcase\bbl@howloaded
2356     \bbl@provide@new{#2}%
2357 \else
2358     \bbl@ifblank{#1}%
2359     {}% With \bbl@load@basic below
2360     {\bbl@provide@renew{#2}}%
2361 \fi
2362 % Post tasks
2363 % -----
2364 % == subsequent calls after the first provide for a locale ==
2365 \ifx\bbl@inidata\@empty\else
2366     \bbl@extend@ini{#2}%
2367 \fi
2368 % == ensure captions ==
2369 \ifx\bbl@KVP@captions\@nnil\else
2370     \bbl@ifunset{bbl@extracaps@#2}%
2371     {\bbl@exp{\\babelensure[exclude=\\today]{#2}}}%
2372     {\bbl@exp{\\babelensure[exclude=\\today,
2373         include=\[bbl@extracaps@#2]]{#2}}}%
2374     \bbl@ifunset{bbl@ensure@language}%
2375     {\bbl@exp{%
2376         \\DeclareRobustCommand\<bbl@ensure@language>[1]{%
2377             \\foreignlanguage{language}%
2378             {###1}}}%
2379     }%
2380     \bbl@exp{%
2381         \\bbl@tglobal\<bbl@ensure@language>%
2382         \\bbl@tglobal\<bbl@ensure@language\space>%
2383     \fi
2384 % ==
2385 % At this point all parameters are defined if 'import'. Now we
2386 % execute some code depending on them. But what about if nothing was
2387 % imported? We just set the basic parameters, but still loading the
2388 % whole ini file.
2389 \bbl@load@basic{#2}%
2390 % == script, language ==
2391 % Override the values from ini or defines them
2392 \ifx\bbl@KVP@script\@nnil\else
2393     \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2394 \fi
2395 \ifx\bbl@KVP@language\@nnil\else
2396     \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2397 \fi
2398 \ifcase\bbl@engine\or
2399     \bbl@ifunset{bbl@chrng@language}{}%
2400     {\directlua{
2401         Babel.set_chranges_b('\bbl@cl{sbc}', '\bbl@cl{chrng}') }}%
2402 \fi
2403 % == onchar ==
2404 \ifx\bbl@KVP@onchar\@nnil\else
2405     \bbl@luahyphenate

```

```

2406 \bbl@exp{%
2407   \\\AddToHook{env/document/before}{\select@language{#2}}}%
2408 \directlua{
2409   if Babel.locale_mapped == nil then
2410     Babel.locale_mapped = true
2411     Babel.linebreaking.add_before(Babel.locale_map)
2412     Babel.loc_to_scr = {}
2413     Babel.chr_to_loc = Babel.chr_to_loc or {}
2414   end
2415   Babel.locale_props[\the\localeid].letters = false
2416 }%
2417 \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
2418 \ifin@
2419   \directlua{
2420     Babel.locale_props[\the\localeid].letters = true
2421   }%
2422 \fi
2423 \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2424 \ifin@
2425   \ifx\bbl@starthyphens\undefined % Needed if no explicit selection
2426     \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
2427   \fi
2428   \bbl@exp{\bbl@add\bbl@starthyphens
2429     {\bbl@patterns@lua{\language}}}%
2430   % TODO - error/warning if no script
2431   \directlua{
2432     if Babel.script_blocks['\bbl@cl{sbc}'] then
2433       Babel.loc_to_scr[\the\localeid] =
2434         Babel.script_blocks['\bbl@cl{sbc}']
2435       Babel.locale_props[\the\localeid].lc = \the\localeid\space
2436       Babel.locale_props[\the\localeid].lg = \the\nameuse{l@\language}\space
2437     end
2438   }%
2439 \fi
2440 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2441 \ifin@
2442   \bbl@ifunset{bbl@lsys@\language}{\bbl@provide@lsys{\language}}}%
2443   \bbl@ifunset{bbl@wdir@\language}{\bbl@provide@dirs{\language}}}%
2444   \directlua{
2445     if Babel.script_blocks['\bbl@cl{sbc}'] then
2446       Babel.loc_to_scr[\the\localeid] =
2447         Babel.script_blocks['\bbl@cl{sbc}']
2448     end}%
2449   \ifx\bbl@mapselect\undefined % TODO. almost the same as mapfont
2450     \AtBeginDocument{%
2451       \bbl@patchfont{\bbl@mapselect}%
2452       {\selectfont}%
2453       \def\bbl@mapselect{%
2454         \let\bbl@mapselect\relax
2455         \edef\bbl@prefontid{\fontid\font}%
2456         \def\bbl@mapdir##1{%
2457           {\def\language{##1}%
2458             \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2459             \bbl@switchfont
2460             \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2461               \directlua{
2462                 Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
2463                   ['\bbl@prefontid'] = \fontid\font\space}%
2464             \fi}}%
2465           \fi
2466       \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
2467 \fi
2468 % TODO - catch non-valid values

```

```

2469 \fi
2470 % == mapfont ==
2471 % For bidi texts, to switch the font based on direction
2472 \ifx\bbbl@KVP@mapfont\@nnil\else
2473   \bbbl@ifsamestring{\bbbl@KVP@mapfont}{direction}}}%
2474   {\bbbl@error{Option '\bbbl@KVP@mapfont' unknown for\%
2475     mapfont. Use 'direction'.%
2476     {See the manual for details.}}}%
2477   \bbbl@ifunset{\bbbl@sys\language\language}{\bbbl@provide@sys\language}}}%
2478   \bbbl@ifunset{\bbbl@wdir\language\language}{\bbbl@provide@dirs\language}}}%
2479   \ifx\bbbl@mapselect\@undefined % TODO. See onchar.
2480     \AtBeginDocument{%
2481       \bbbl@patchfont{\bbbl@mapselect}}}%
2482     {\selectfont}}}%
2483     \def\bbbl@mapselect{%
2484       \let\bbbl@mapselect\relax
2485       \edef\bbbl@prefontid{\fontid\font}}%
2486     \def\bbbl@mapdir##1{%
2487       {\def\language{##1}%
2488       \let\bbbl@ifrestoring\@firstoftwo % avoid font warning
2489       \bbbl@switchfont
2490       \directlua{Babel.fontmap
2491         [\the\csname \bbbl@wdir@##1\endcsname]%
2492         [\bbbl@prefontid]=\fontid\font}}}%
2493   \fi
2494   \bbbl@exp{\bbbl@add\bbbl@mapselect{\bbbl@mapdir\language}}}%
2495 \fi
2496 % == Line breaking: intraspace, intrapenalty ==
2497 % For CJK, East Asian, Southeast Asian, if interspace in ini
2498 \ifx\bbbl@KVP@intraspace\@nnil\else % We can override the ini or set
2499   \bbbl@csarg\edef{intsp@#2}{\bbbl@KVP@intraspace}%
2500 \fi
2501 \bbbl@provide@intraspace
2502 % == Line breaking: CJK quotes ==
2503 \ifcase\bbbl@engine\or
2504   \bbbl@xin@{/c}{/\bbbl@cl{lbrk}}}%
2505   \ifin@
2506     \bbbl@ifunset{\bbbl@quote@\language}}}%
2507     {\directlua{
2508       Babel.locale_props[\the\localeid].cjk_quotes = {}
2509       local cs = 'op'
2510       for c in string.utfvalues(
2511         [[\csname \bbbl@quote@\language\endcsname]]) do
2512         if Babel.cjk_characters[c].c == 'qu' then
2513           Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2514         end
2515         cs = ( cs == 'op') and 'cl' or 'op'
2516       end
2517     }}%
2518   \fi
2519 \fi
2520 % == Line breaking: justification ==
2521 \ifx\bbbl@KVP@justification\@nnil\else
2522   \let\bbbl@KVP@linebreaking\bbbl@KVP@justification
2523 \fi
2524 \ifx\bbbl@KVP@linebreaking\@nnil\else
2525   \bbbl@xin@{\bbbl@KVP@linebreaking,}%
2526   {\elongated,kashida,cjk,padding,unhyphenated,}%
2527   \ifin@
2528     \bbbl@csarg\xdef
2529     {\lbrk@\language}{\expandafter\@car\bbbl@KVP@linebreaking\@nil}%
2530   \fi
2531 \fi

```

```

2532 \bbl@xin@{/e}{/\bbl@cl{lbrk}}%
2533 \ifin@else\bbl@xin@{/k}{/\bbl@cl{lbrk}}\fi
2534 \ifin@\bbl@arabicjust\fi
2535 \bbl@xin@{/p}{/\bbl@cl{lbrk}}%
2536 \ifin@AtBeginDocument{\@nameuse{bbl@tibetanjust}}\fi
2537 % == Line breaking: hyphenate.other.(locale|script) ==
2538 \ifx\bbl@lbrkflag\empty
2539   \bbl@ifunset{bbl@hyotl@language}{}%
2540   {\bbl@csarg\bbl@replace{hyotl@language}{ }{,}%
2541     \bbl@startcommands*\language}%
2542   \bbl@csarg\bbl@foreach{hyotl@language}{%
2543     \ifcase\bbl@engine
2544       \ifnum##1<257
2545         \SetHyphenMap{\BabelLower{##1}{##1}}%
2546       \fi
2547     \else
2548       \SetHyphenMap{\BabelLower{##1}{##1}}%
2549     \fi}%
2550   \bbl@endcommands%
2551 \bbl@ifunset{bbl@hyots@language}{}%
2552 {\bbl@csarg\bbl@replace{hyots@language}{ }{,}%
2553   \bbl@csarg\bbl@foreach{hyots@language}{%
2554     \ifcase\bbl@engine
2555       \ifnum##1<257
2556         \global\lccode##1=##1\relax
2557       \fi
2558     \else
2559       \global\lccode##1=##1\relax
2560     \fi}}%
2561 \fi
2562 % == Counters: maparabic ==
2563 % Native digits, if provided in ini (TeX level, xe and lua)
2564 \ifcase\bbl@engine\else
2565   \bbl@ifunset{bbl@dgnat@language}{}%
2566   {\expandafter\ifx\csname bbl@dgnat@language\endcsname\@empty\else
2567     \expandafter\expandafter\expandafter
2568     \bbl@setdigits\csname bbl@dgnat@language\endcsname
2569     \ifx\bbl@KVP@maparabic\@nnil\else
2570       \ifx\bbl@latinarabic\undefined
2571         \expandafter\let\expandafter\@arabic
2572         \csname bbl@counter@language\endcsname
2573       \else % ie, if layout=counters, which redefines \@arabic
2574         \expandafter\let\expandafter\bbl@latinarabic
2575         \csname bbl@counter@language\endcsname
2576       \fi
2577     \fi
2578   \fi}%
2579 \fi
2580 % == Counters: mapdigits ==
2581 % Native digits (lua level).
2582 \ifodd\bbl@engine
2583   \ifx\bbl@KVP@mapdigits\@nnil\else
2584     \bbl@ifunset{bbl@dgnat@language}{}%
2585     {\RequirePackage{luatexbase}%
2586       \bbl@activate@preotf
2587       \directlua{
2588         Babel = Babel or {} %%% -> presets in luababel
2589         Babel.digits_mapped = true
2590         Babel.digits = Babel.digits or {}
2591         Babel.digits[\the\localeid] =
2592           table.pack(string.utfvalue('\bbl@cl{dgnat}'))
2593         if not Babel.numbers then
2594           function Babel.numbers(head)

```

```

2595         local LOCALE = Babel.attr_locale
2596         local GLYPH = node.id'glyph'
2597         local inmath = false
2598         for item in node.traverse(head) do
2599             if not inmath and item.id == GLYPH then
2600                 local temp = node.get_attribute(item, LOCALE)
2601                 if Babel.digits[temp] then
2602                     local chr = item.char
2603                     if chr > 47 and chr < 58 then
2604                         item.char = Babel.digits[temp][chr-47]
2605                     end
2606                 end
2607             elseif item.id == node.id'math' then
2608                 inmath = (item.subtype == 0)
2609             end
2610         end
2611         return head
2612     end
2613 end
2614 }}%
2615 \fi
2616 \fi
2617 % == Counters: alph, Alph ==
2618 % What if extras<lang> contains a \babel@save\@alph? It won't be
2619 % restored correctly when exiting the language, so we ignore
2620 % this change with the \bbl@alph@saved trick.
2621 \ifx\bbl@KVP@alph\@nnil\else
2622     \bbl@extras@wrap{\bbl@alph@saved}%
2623     {\let\bbl@alph@saved\@alph}%
2624     {\let\@alph\bbl@alph@saved
2625      \babel@save\@alph}%
2626     \bbl@exp{%
2627         \bbl@add\<extras\languagename>{%
2628             \let\@alph\bbl@cntr\bbl@KVP@alph @\languagename>}}%
2629 \fi
2630 \ifx\bbl@KVP@Alph\@nnil\else
2631     \bbl@extras@wrap{\bbl@Alph@saved}%
2632     {\let\bbl@Alph@saved\@Alph}%
2633     {\let\@Alph\bbl@Alph@saved
2634      \babel@save\@Alph}%
2635     \bbl@exp{%
2636         \bbl@add\<extras\languagename>{%
2637             \let\@Alph\bbl@cntr\bbl@KVP@Alph @\languagename>}}%
2638 \fi
2639 % == Calendars ==
2640 \ifx\bbl@KVP@calendar\@nnil
2641     \edef\bbl@KVP@calendar{\bbl@cl{calpr}}}%
2642 \fi
2643 \def\bbl@tempe##1 ##2\@{% % Get first calendar
2644     \def\bbl@tempa{##1}}%
2645     \bbl@exp{\bbl@tempe\bbl@KVP@calendar\space\@}%
2646 \def\bbl@tempe##1.##2.##3\@{%
2647     \def\bbl@tempc{##1}%
2648     \def\bbl@tempb{##2}}%
2649 \expandafter\bbl@tempe\bbl@tempa..\@
2650 \bbl@csarg\edef{calpr@\languagename}{%
2651     \ifx\bbl@tempc\@empty\else
2652         calendar=\bbl@tempc
2653     \fi
2654     \ifx\bbl@tempb\@empty\else
2655         ,variant=\bbl@tempb
2656     \fi}%
2657 % == require.babel in ini ==

```

```

2658 % To load or reload the babel-*.tex, if require.babel in ini
2659 \ifx\bbbl@beforestart\relax\else % But not in doc aux or body
2660   \bbbl@ifunset{\bbbl@rqtex@\language}\relax\else
2661     {\expandafter\ifx\csname \bbbl@rqtex@\language\endcsname\@empty\else
2662       \let\BabelBeforeIni\@gobbles
2663       \chardef\atcatcode=\catcode\@
2664       \catcode\@=11\relax
2665       \bbbl@input\@texini{\bbbl@cs{rqtex@\language}}%
2666       \catcode\@=\atcatcode
2667       \let\atcatcode\relax
2668       \global\bbbl@csarg\let{rqtex@\language}\relax
2669     \fi}%
2670 \bbbl@foreach\bbbl@calendars{%
2671   \bbbl@ifunset{\bbbl@ca##1}{%
2672     \chardef\atcatcode=\catcode\@
2673     \catcode\@=11\relax
2674     \InputIfFileExists{babel-ca-##1.tex}{\relax}%
2675     \catcode\@=\atcatcode
2676     \let\atcatcode\relax}%
2677   }%
2678 \fi
2679 % == frenchspacing ==
2680 \ifcase\bbbl@howloaded\in@true\else\in@false\fi
2681 \ifin@ \else\bbbl@x@{\typography/frenchspacing}{\bbbl@key@list}\fi
2682 \ifin@
2683   \bbbl@extras@wrap{\bbbl@pre@fs}%
2684   {\bbbl@pre@fs}%
2685   {\bbbl@post@fs}%
2686 \fi
2687 % == transforms ==
2688 \ifodd\bbbl@engine
2689   \ifx\bbbl@KVP@transforms\@nnil\else
2690     \def\bbbl@elt##1##2##3{%
2691       \in@{\$transforms.}{\$##1}%
2692       \ifin@
2693         \def\bbbl@tempa{##1}%
2694         \bbbl@replace\bbbl@tempa{transforms.}{}%
2695         \bbbl@carg\bbbl@transforms{babel\bbbl@tempa}{##2}{##3}%
2696       \fi}%
2697     \csname \bbbl@inidata@\language\endcsname
2698     \bbbl@release@transforms\relax % \relax closes the last item.
2699   \fi
2700 \fi
2701 % == main ==
2702 \ifx\bbbl@KVP@main\@nnil % Restore only if not 'main'
2703   \let\language\bbbl@savelanguage
2704   \chardef\localeid\bbbl@savelocaleid\relax
2705 \fi}

```

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

```

2706 \def\bbbl@provide@new#1{%
2707   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2708   \@namedef{extras#1}{}%
2709   \@namedef{noextras#1}{}%
2710   \bbbl@startcommands*{#1}{captions}%
2711   \ifx\bbbl@KVP@captions\@nnil % and also if import, implicit
2712     \def\bbbl@tempb##1{% elt for \bbbl@captionslist
2713       \ifx##1\@empty\else
2714         \bbbl@exp{%
2715           \SetString\##1{%
2716             \bbbl@nocaption{\bbbl@stripslash##1}{#1\bbbl@stripslash##1}}%
2717           \expandafter\bbbl@tempb

```



```

2718     \fi}%
2719     \expandafter\bbbl@tempb\bbbl@captionslist\@empty
2720 \else
2721     \ifx\bbbl@initoload\relax
2722         \bbbl@read@ini{\bbbl@KVP@captions}2%    % Here letters cat = 11
2723     \else
2724         \bbbl@read@ini{\bbbl@initoload}2%        % Same
2725     \fi
2726 \fi
2727 \StartBabelCommands*{#1}{date}%
2728 \ifx\bbbl@KVP@date\@nnil
2729     \bbbl@exp{%
2730         \SetString\@today{\bbbl@nocaption{today}{#1today}}}%
2731 \else
2732     \bbbl@savetoday
2733     \bbbl@savedate
2734 \fi
2735 \bbbl@endcommands
2736 \bbbl@load@basic{#1}%
2737 % == hyphenmins == (only if new)
2738 \bbbl@exp{%
2739     \gdef\<#1hyphenmins>{%
2740         {\bbbl@ifunset{\bbbl@lfthm@#1}{2}{\bbbl@cs{lfthm@#1}}}%
2741         {\bbbl@ifunset{\bbbl@rgthm@#1}{3}{\bbbl@cs{rgthm@#1}}}}}%
2742 % == hyphenrules (also in renew) ==
2743 \bbbl@provide@hyphens{#1}%
2744 \ifx\bbbl@KVP@main\@nnil\else
2745     \expandafter\main@language\expandafter{#1}%
2746 \fi}
2747 %
2748 \def\bbbl@provide@renew#1{%
2749     \ifx\bbbl@KVP@captions\@nnil\else
2750         \StartBabelCommands*{#1}{captions}%
2751         \bbbl@read@ini{\bbbl@KVP@captions}2%    % Here all letters cat = 11
2752         \EndBabelCommands
2753     \fi
2754     \ifx\bbbl@KVP@date\@nnil\else
2755         \StartBabelCommands*{#1}{date}%
2756         \bbbl@savetoday
2757         \bbbl@savedate
2758         \EndBabelCommands
2759     \fi
2760 % == hyphenrules (also in new) ==
2761 \ifx\bbbl@lbfkflag\@empty
2762     \bbbl@provide@hyphens{#1}%
2763 \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.)

```

2764 \def\bbbl@load@basic#1{%
2765     \ifcase\bbbl@howloaded\or\or
2766         \ifcase\csname bbl@llevel\language\endcsname
2767             \bbbl@csarg\let{lname@\language}\relax
2768         \fi
2769     \fi
2770     \bbbl@ifunset{\bbbl@lname@#1}%
2771     {\def\BabelBeforeIni##1##2{%
2772         \begingroup
2773         \let\bbbl@ini@captions@aux\@gobbletwo
2774         \def\bbbl@inidate #####1.####2.####3.####4\relax #####5####6}%
2775         \bbbl@read@ini{##1}%
2776         \ifx\bbbl@initoload\relax\endinput\fi

```

```

2777     \endgroup}%
2778     \begingroup      % boxed, to avoid extra spaces:
2779     \ifx\bbbl@initoload\relax
2780     \bbbl@input@texini{#1}%
2781     \else
2782     \setbox\z@\hbox{\BabelBeforeIni{\bbbl@initoload}}}%
2783     \fi
2784     \endgroup}%
2785     {}%

```

The hyphenrules option is handled with an auxiliary macro.

```

2786 \def\bbbl@provide@hyphens#1{%
2787   \let\bbbl@tempa\relax
2788   \ifx\bbbl@KVP@hyphenrules\@nnil\else
2789     \bbbl@replace\bbbl@KVP@hyphenrules{ }{,}%
2790     \bbbl@foreach\bbbl@KVP@hyphenrules{%
2791       \ifx\bbbl@tempa\relax      % if not yet found
2792       \bbbl@ifsamestring{##1}{+}%
2793       {\bbbl@exp{\addlanguage<#1>}}}%
2794       {}%
2795       \bbbl@ifunset{#1}%
2796       {}%
2797       {\bbbl@exp{\let\bbbl@tempa<#1>}}}%
2798     \fi%
2799     \ifx\bbbl@tempa\relax
2800     \bbbl@warning{%
2801       Requested 'hyphenrules=' for '\language' not found.\\
2802       Using the default value. Reported}%
2803     \fi
2804   \fi
2805   \ifx\bbbl@tempa\relax %          if no opt or no language in opt found
2806     \ifx\bbbl@KVP@import\@nnil
2807       \ifx\bbbl@initoload\relax\else
2808         \bbbl@exp{%
2809           and hyphenrules is not empty
2810           \\bbbl@ifblank{\bbbl@cs{hyphr@#1}}}%
2811           {}%
2812           {\let\\bbbl@tempa<#1>\bbbl@cl{hyphr}>}}}%
2813       \fi
2814     \else % if importing
2815       \bbbl@exp{%
2816         and hyphenrules is not empty
2817         \\bbbl@ifblank{\bbbl@cs{hyphr@#1}}}%
2818         {}%
2819         {\let\\bbbl@tempa<#1>\bbbl@cl{hyphr}>}}}%
2820     \fi
2821     \bbbl@ifunset{\bbbl@tempa}%      ie, relax or undefined
2822     {\bbbl@ifunset{#1}%              no hyphenrules found - fallback
2823     {\bbbl@exp{\adddialect<#1>\language}}}%
2824     {}%
2825     {\bbbl@exp{\adddialect<#1>\bbbl@tempa}}}% found in opt list or ini

```

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

```

2825 \def\bbbl@input@texini#1{%
2826   \bbbl@bsphack
2827   \bbbl@exp{%
2828     \catcode`\=14 \catcode`\=0
2829     \catcode`\={1} \catcode`\={2
2830     \lowercase{\InputIfFileExists{babel-#1.tex}}}%
2831     \catcode`\=\\the\catcode`\=\\relax
2832     \catcode`\={\\the\catcode`\={\\relax
2833     \catcode`\={\\the\catcode`\={\\relax
2834     \catcode`\={\\the\catcode`\={\\relax}%
2835   \bbbl@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.

```

2836 \def\bbl@inline#1\bbl@inline{%
2837   \@ifnextchar[\bbl@inisect{\@ifnextchar\bbl@iniskip\bbl@inistore}#1\@}% ]
2838 \def\bbl@inisect[#1]#2\@{\def\bbl@section{#1}}
2839 \def\bbl@iniskip#1\@{\%      if starts with ;
2840 \def\bbl@inistore#1=#2\@{\%      full (default)
2841   \bbl@trim@def\bbl@tempa{#1}%
2842   \bbl@trim\toks@{#2}%
2843   \bbl@xin@{;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2844   \ifin@ \else
2845     \bbl@xin@{,identification/include.}%
2846     {,\bbl@section/\bbl@tempa}%
2847   \ifin@ \edef\bbl@required@inis{\the\toks@} \fi
2848   \bbl@exp{%
2849     \\g@addto@macro\\bbl@inidata{%
2850       \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2851   \fi}
2852 \def\bbl@inistore@min#1=#2\@{\%  minimal (maybe set in \bbl@read@ini)
2853   \bbl@trim@def\bbl@tempa{#1}%
2854   \bbl@trim\toks@{#2}%
2855   \bbl@xin@{.identification.}{.\bbl@section.}%
2856   \ifin@
2857     \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2858       \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2859   \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.

```

2860 \def\bbl@loop@ini{%
2861   \loop
2862     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2863     \endlinechar\m@ne
2864     \read\bbl@readstream to \bbl@line
2865     \endlinechar``^^M
2866     \ifx\bbl@line\empty\else
2867       \expandafter\bbl@inline\bbl@line\bbl@inline
2868     \fi
2869     \repeat}
2870 \ifx\bbl@readstream\undefined
2871   \csname newread\endcsname\bbl@readstream
2872 \fi
2873 \def\bbl@read@ini#1#2{%
2874   \global\let\bbl@extend@ini\gobble
2875   \openin\bbl@readstream=babel-#1.ini
2876   \ifeof\bbl@readstream
2877     \bbl@error
2878     {There is no ini file for the requested language\\%
2879     (#1: \language). Perhaps you misspelled it or your\\%
2880     installation is not complete.}%
2881     {Fix the name or reinstall babel.}%
2882   \else
2883     % == Store ini data in \bbl@inidata ==
2884     \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2885     \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2886     \bbl@info{Importing
2887       \ifcase#2font and identification \or basic \fi
2888       data for \language\\%

```

```

2889         from babel-#1.ini. Reported}%
2890 \ifnum#2=\z@
2891     \global\let\bbl@inidata\@empty
2892     \let\bbl@inistore\bbl@inistore@min    % Remember it's local
2893 \fi
2894 \def\bbl@section{identification}%
2895 \let\bbl@required@inis\@empty
2896 \bbl@exp{\bbl@inistore tag.ini=#1\\@}%
2897 \bbl@inistore load.level=#2\@@
2898 \bbl@loop@ini
2899 \ifx\bbl@required@inis\@empty\else
2900     \bbl@replace\bbl@required@inis{ },}%
2901     \bbl@foreach\bbl@required@inis{%
2902         \openin\bbl@readstream=##1.ini
2903         \bbl@loop@ini}%
2904 \fi
2905 % == Process stored data ==
2906 \bbl@csarg\xdef{lini@\languagename}{#1}%
2907 \bbl@read@ini@aux
2908 % == 'Export' data ==
2909 \bbl@ini@exports{#2}%
2910 \global\bbl@csarg\let{inidata@\languagename}\bbl@inidata
2911 \global\let\bbl@inidata\@empty
2912 \bbl@exp{\bbl@add@list\bbl@ini@loaded{\languagename}}%
2913 \bbl@tglobal\bbl@ini@loaded
2914 \fi}
2915 \def\bbl@read@ini@aux{%
2916     \let\bbl@savestrings\@empty
2917     \let\bbl@savetoday\@empty
2918     \let\bbl@savestate\@empty
2919     \def\bbl@elt##1##2##3{%
2920         \def\bbl@section{##1}%
2921         \in@{=date.}{=##1}% Find a better place
2922         \ifin@
2923             \bbl@ifunset{bbl@inikv@##1}%
2924             {\bbl@ini@calendar{##1}}%
2925             {}%
2926         \fi
2927         \in@{=identification/extension.}{=##1/##2}%
2928         \ifin@
2929             \bbl@ini@extension{##2}%
2930         \fi
2931         \bbl@ifunset{bbl@inikv@##1}{}%
2932         {\csname bbl@inikv@##1\endcsname{##2}{##3}}%
2933     \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.

```

2934 \def\bbl@extend@ini@aux#1{%
2935     \bbl@startcommands*{#1}{captions}%
2936     % Activate captions/... and modify exports
2937     \bbl@csarg\def{inikv@captions.licr}##1##2{%
2938         \setlocalecaption{#1}{##1}{##2}}%
2939     \def\bbl@inikv@captions##1##2{%
2940         \bbl@ini@captions@aux{##1}{##2}}%
2941     \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2942     \def\bbl@exportkey##1##2##3{%
2943         \bbl@ifunset{bbl@kv@##2}{%
2944             {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2945                 \bbl@exp{\global\let<bbl@##1@\languagename>\<bbl@kv@##2>}}%
2946             \fi}}%
2947     % As with \bbl@read@ini, but with some changes
2948     \bbl@read@ini@aux

```

```

2949 \bbl@ini@exports\tw@
2950 % Update inidata@lang by pretending the ini is read.
2951 \def\bbl@elt##1##2##3{%
2952     \def\bbl@section{##1}%
2953     \bbl@inline##2=##3\bbl@inline}%
2954 \csname bbl@inidata@#1\endcsname
2955 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2956 \StartBabelCommands*{#1}{date}% And from the import stuff
2957 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2958 \bbl@savetoday
2959 \bbl@savedate
2960 \bbl@endcommands}

```

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

```

2961 \def\bbl@ini@calendar#1{%
2962     \lowercase{\def\bbl@tempa{= #1=}}%
2963     \bbl@replace\bbl@tempa{=date.gregorian}{}%
2964     \bbl@replace\bbl@tempa{=date.}{}%
2965     \in@{.licr=}{#1=}%
2966     \ifin@
2967         \ifcase\bbl@engine
2968             \bbl@replace\bbl@tempa{.licr=}{}%
2969         \else
2970             \let\bbl@tempa\relax
2971         \fi
2972     \fi
2973     \ifx\bbl@tempa\relax\else
2974         \bbl@replace\bbl@tempa{=}{}%
2975         \ifx\bbl@tempa\@empty\else
2976             \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2977         \fi
2978         \bbl@exp{%
2979             \def\<bbl@inikv@#1>####1####2{%
2980                 \bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2981     \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).

```

2982 \def\bbl@renewinikey#1/#2\@#3{%
2983     \edef\bbl@tempa{\zap@space #1 \@empty}% section
2984     \edef\bbl@tempb{\zap@space #2 \@empty}% key
2985     \bbl@trim\toks@{#3}% value
2986     \bbl@exp{%
2987         \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2988         \\g@addto@macro\\bbl@inidata{%
2989             \\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.

```

2990 \def\bbl@exportkey#1#2#3{%
2991     \bbl@ifunset{bbl@kv@#2}%
2992     {\bbl@csarg\gdef{#1@\language}{#3}}%
2993     {\expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
2994         \bbl@csarg\gdef{#1@\language}{#3}}%
2995     \else
2996         \bbl@exp{\global\let\<bbl@#1@\language>\<bbl@kv@#2>}%
2997     \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.

```

2998 \def\bbl@iniwarning#1{%

```

```

2999 \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
3000 {\bbl@warning{%
3001     From babel-\bbl@cs{lini@language}.ini:\%
3002     \bbl@cs{@kv@identification.warning#1}\%
3003     Reported }}}
3004 %
3005 \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.

```

3006 \def\bbl@ini@extension#1{%
3007   \def\bbl@tempa{#1}%
3008   \bbl@replace\bbl@tempa{extension.}{}%
3009   \bbl@replace\bbl@tempa{.tag.bcp47}{}%
3010   \bbl@ifunset{\bbl@info@#1}%
3011   {\bbl@csarg\xdef{info@#1}{ext/\bbl@tempa}%
3012    \bbl@exp{%
3013      \\g@addto@macro\\bbl@moreinfo{%
3014        \\bbl@exportkey{ext/\bbl@tempa}{identification.#1}{}}}%
3015   {}}
3016 \let\bbl@moreinfo\@empty
3017 %
3018 \def\bbl@ini@exports#1{%
3019   % Identification always exported
3020   \bbl@iniwarning{}%
3021   \ifcase\bbl@engine
3022     \bbl@iniwarning{.pdflatex}%
3023   \or
3024     \bbl@iniwarning{.luaLatex}%
3025   \or
3026     \bbl@iniwarning{.xetex}%
3027   \fi%
3028   \bbl@exportkey{llevel}{identification.load.level}{}%
3029   \bbl@exportkey{elname}{identification.name.english}{}%
3030   \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%
3031     {\csname bbl@elname@language\endcsname}}%
3032   \bbl@exportkey{tbc}{identification.tag.bcp47}{}%
3033   \bbl@exportkey{lbc}{identification.language.tag.bcp47}{}%
3034   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3035   \bbl@exportkey{esname}{identification.script.name}{}%
3036   \bbl@exp{\\bbl@exportkey{sname}{identification.script.name.opentype}%
3037     {\csname bbl@esname@language\endcsname}}%
3038   \bbl@exportkey{sbc}{identification.script.tag.bcp47}{}%
3039   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3040   \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
3041   \bbl@exportkey{vbc}{identification.variant.tag.bcp47}{}%
3042   \bbl@moreinfo
3043   % Also maps bcp47 -> language
3044   \ifbbl@bcptoname
3045     \bbl@csarg\xdef{bcp@map@bbl@cl{tbc}}{\language}%
3046   \fi
3047   % Conditional
3048   \ifnum#1>\z@ % 0 = only info, 1, 2 = basic, (re)new
3049     \bbl@exportkey{calpr}{date.calendar.preferred}{}%
3050     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3051     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3052     \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
3053     \bbl@exportkey{rgtm}{typography.righthyphenmin}{3}%
3054     \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3055     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3056     \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3057     \bbl@exportkey{intsp}{typography.intraspaces}{}%
3058     \bbl@exportkey{frspc}{typography.frenchspacing}{u}%

```

```

3059 \bbl@exportkey{chrng}{characters.ranges}{}%
3060 \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
3061 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3062 \ifnum#1=\tw@ % only (re)new
3063 \bbl@exportkey{rqtex}{identification.require.babel}{}%
3064 \bbl@tglobal\bbl@savetoday
3065 \bbl@tglobal\bbl@savedate
3066 \bbl@savestrings
3067 \fi
3068 \fi}

```

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

```

3069 \def\bbl@inikv#1#2{%      key=value
3070 \toks@{#2}%              This hides #'s from ini values
3071 \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}

```

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

```

3072 \let\bbl@inikv@identification\bbl@inikv
3073 \let\bbl@inikv@date\bbl@inikv
3074 \let\bbl@inikv@typography\bbl@inikv
3075 \let\bbl@inikv@characters\bbl@inikv
3076 \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 \localnumeral, and another one preserving the trailing .1 for the ‘units’.

```

3077 \def\bbl@inikv@counters#1#2{%
3078 \bbl@ifsamestring{#1}{digits}%
3079 {\bbl@error{The counter name 'digits' is reserved for mapping\%
3080 decimal digits}%
3081 {Use another name.}}%
3082 {}%
3083 \def\bbl@tempc{#1}%
3084 \bbl@trim@def{\bbl@tempb*}{#2}%
3085 \in@{.1$}{#1$}%
3086 \ifin@
3087 \bbl@replace\bbl@tempc{.1}{}%
3088 \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}%
3089 \noexpand\bbl@alphanumeric{\bbl@tempc}%
3090 \fi
3091 \in@{.F.}{#1}%
3092 \ifin@ \else \in@{.S.}{#1} \fi
3093 \ifin@
3094 \bbl@csarg\protected@xdef{cntr@#1@\language}{\bbl@tempb*}%
3095 \else
3096 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3097 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3098 \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
3099 \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.

```

3100 \ifcase\bbl@engine
3101 \bbl@csarg\def{inikv@captions.licr}#1#2{%
3102 \bbl@ini@captions@aux{#1}{#2}}
3103 \else
3104 \def\bbl@inikv@captions#1#2{%
3105 \bbl@ini@captions@aux{#1}{#2}}
3106 \fi

```

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

```

3107 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3108 \bbl@replace\bbl@tempa{.template}{}%

```

```

3109 \def\bbl@toreplace{#1{}}%
3110 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3111 \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3112 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3113 \bbl@replace\bbl@toreplace{[ ]}{\name\endcsname{}}}%
3114 \bbl@replace\bbl@toreplace{[ ]}{\endcsname{}}}%
3115 \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3116 \ifin@
3117 \@nameuse{\bbl@patch\bbl@tempa}%
3118 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3119 \fi
3120 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3121 \ifin@
3122 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3123 \bbl@exp{\gdef<\fnum@\bbl@tempa>{%
3124 \\\bbl@ifunset{\bbl@\bbl@tempa fmt@\\language}%
3125 {[fnum@\bbl@tempa]}%
3126 {\\\@nameuse{\bbl@\bbl@tempa fmt@\\language}}}%
3127 \fi}
3128 \def\bbl@ini@captions@aux#1#2{%
3129 \bbl@trim\def\bbl@tempa{#1}%
3130 \bbl@xin@{.template}{\bbl@tempa}%
3131 \ifin@
3132 \bbl@ini@captions@template{#2}\language
3133 \else
3134 \bbl@ifblank{#2}%
3135 {\bbl@exp{%
3136 \toks@{\\\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
3137 {\bbl@trim\toks@{#2}}%
3138 \bbl@exp{%
3139 \\\bbl@add\\bbl@savestrings{%
3140 \\\SetString<\bbl@tempa name>{\the\toks@}}%
3141 \toks@\expandafter{\bbl@captionslist}%
3142 \bbl@exp{\\\in@{\<\bbl@tempa name>}{\the\toks@}}%
3143 \ifin@else
3144 \bbl@exp{%
3145 \\\bbl@add<\bbl@extracaps@\language>{\<\bbl@tempa name>}%
3146 \\\bbl@tglobal<\bbl@extracaps@\language>}%
3147 \fi
3148 \fi}

```

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

```

3149 \def\bbl@list@the{%
3150 part,chapter,section,subsection,subsubsection,paragraph,%
3151 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3152 table,page,footnote,mpfootnote,mpfn}
3153 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3154 \bbl@ifunset{\bbl@map@#1@\language}%
3155 {\@nameuse{#1}}%
3156 {\@nameuse{\bbl@map@#1@\language}}}
3157 \def\bbl@inikv@labels#1#2{%
3158 \in@{.map}{#1}%
3159 \ifin@
3160 \ifx\bbl@KVP@labels\@nnil\else
3161 \bbl@xin@{ map }{\bbl@KVP@labels\space}%
3162 \ifin@
3163 \def\bbl@tempc{#1}%
3164 \bbl@replace\bbl@tempc{.map}{}%
3165 \in@{,#2,}{,arabic,roman,Roman,alpha,Alpha,fnsymbol,}%
3166 \bbl@exp{%
3167 \gdef<\bbl@map@\bbl@tempc @\language>%
3168 {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3169 \bbl@foreach\bbl@list@the{%

```



```

3170 \bbl@ifunset{the##1}{}%
3171 {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3172 \bbl@exp{%
3173 \\bbl@sreplace\<the##1>%
3174 {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3175 \\bbl@sreplace\<the##1>%
3176 {\<\@empty @\bbl@tempc>\<c@##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3177 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3178 \toks@\expandafter\expandafter\expandafter{%
3179 \csname the##1\endcsname}%
3180 \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3181 \fi}}%
3182 \fi
3183 \fi
3184 %
3185 \else
3186 %
3187 % The following code is still under study. You can test it and make
3188 % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3189 % language dependent.
3190 \in@{enumerate.}{#1}%
3191 \ifin@
3192 \def\bbl@tempa{#1}%
3193 \bbl@replace\bbl@tempa{enumerate.}{}%
3194 \def\bbl@toreplace{#2}%
3195 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3196 \bbl@replace\bbl@toreplace{[]}{\csname the}%
3197 \bbl@replace\bbl@toreplace{[]}{\endcsname{}}%
3198 \toks@\expandafter{\bbl@toreplace}%
3199 % TODO. Execute only once:
3200 \bbl@exp{%
3201 \\bbl@add\<extras\language>{%
3202 \\bbl@save\<labelenum\romannumeral\bbl@tempa>%
3203 \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3204 \\bbl@toggle\<extras\language>}%
3205 \fi
3206 \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.

```

3207 \def\bbl@chapttype{chapter}
3208 \ifx\@makechapterhead\@undefined
3209 \let\bbl@patchchapter\relax
3210 \else\ifx\thechapter\@undefined
3211 \let\bbl@patchchapter\relax
3212 \else\ifx\ps@headings\@undefined
3213 \let\bbl@patchchapter\relax
3214 \else
3215 \def\bbl@patchchapter{%
3216 \global\let\bbl@patchchapter\relax
3217 \gdef\bbl@chfmt{%
3218 \bbl@ifunset{\bbl@bbl@chapttype fmt@\language}%
3219 {\@chapapp\space\thechapter}
3220 {\@nameuse{\bbl@bbl@chapttype fmt@\language}}}%
3221 \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3222 \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3223 \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3224 \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3225 \bbl@toggle\appendix
3226 \bbl@toggle\ps@headings
3227 \bbl@toggle\chaptermark

```

```

3228 \bbl@tglobal\@makechapterhead}
3229 \let\bbl@patchappendix\bbl@patchchapter
3230 \fi\fi\fi
3231 \ifx\@part\@undefined
3232 \let\bbl@patchpart\relax
3233 \else
3234 \def\bbl@patchpart{%
3235 \global\let\bbl@patchpart\relax
3236 \gdef\bbl@partformat{%
3237 \bbl@ifunset{\bbl@partfmt@\language}%
3238 {\partname\nobreakspace\thepart}
3239 {\@nameuse{\bbl@partfmt@\language}}}%
3240 \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3241 \bbl@tglobal\@part}
3242 \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

```

3243 \let\bbl@calendar\@empty
3244 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3245 \def\bbl@localedate#1#2#3#4{%
3246 \begingroup
3247 \edef\bbl@they{#2}%
3248 \edef\bbl@them{#3}%
3249 \edef\bbl@thed{#4}%
3250 \edef\bbl@tempe{%
3251 \bbl@ifunset{\bbl@calpr@\language}{\bbl@cl{calpr}}{,}%
3252 #1}%
3253 \bbl@replace\bbl@tempe{ }{}%
3254 \bbl@replace\bbl@tempe{CONVERT}{convert=}% Hackish
3255 \bbl@replace\bbl@tempe{convert}{convert=}%
3256 \let\bbl@ld@calendar\@empty
3257 \let\bbl@ld@variant\@empty
3258 \let\bbl@ld@convert\relax
3259 \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ld@##1}{##2}}%
3260 \bbl@foreach\bbl@tempe{\bbl@tempb##1\@}%
3261 \bbl@replace\bbl@ld@calendar{gregorian}{}%
3262 \ifx\bbl@ld@calendar\@empty\else
3263 \ifx\bbl@ld@convert\relax\else
3264 \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3265 {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3266 \fi
3267 \fi
3268 \@nameuse{\bbl@precalendar}% Remove, eg, +, -civil (-ca-islamic)
3269 \edef\bbl@calendar{% Used in \month..., too
3270 \bbl@ld@calendar
3271 \ifx\bbl@ld@variant\@empty\else
3272 .\bbl@ld@variant
3273 \fi}%
3274 \bbl@cased
3275 {\@nameuse{\bbl@date@\language @\bbl@calendar}%
3276 \bbl@they\bbl@them\bbl@thed}%
3277 \endgroup}
3278 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3279 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3280 \bbl@trim@def\bbl@tempa{#1.#2}%
3281 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3282 {\bbl@trim@def\bbl@tempa{#3}%
3283 \bbl@trim\toks@{#5}%
3284 \@temptokena\expandafter{\bbl@savestate}%
3285 \bbl@exp{% Reverse order - in ini last wins
3286 \def\\bbl@savestate{%
3287 \\SetString<month\romannumeral\bbl@tempa#6name>{\the\toks@}%

```

```

3288     \the\temptokena}}}%
3289     {\bbl@ifsamestring{\bbl@tempa}{date.long}%      defined now
3290     {\lowercase{\def\bbl@tempb{#6}}}%
3291     \bbl@trim@def\bbl@toreplace{#5}%
3292     \bbl@TG@date
3293     \global\bbl@csarg\let{date@\language name @\bbl@tempb}\bbl@toreplace
3294     \ifx\bbl@savetoday@empty
3295     \bbl@exp{% TODO. Move to a better place.
3296     \\AfterBabelCommands{%
3297     \def<\language name date>{\protect<\language name date >}%
3298     \\newcommand<\language name date >[4][]{%
3299     \\bbl@usedategroupttrue
3300     <\bbl@ensure@\language name>{%
3301     \\localedate[####1]{####2}{####3}{####4}}}%
3302     \def\\bbl@savetoday{%
3303     \\SetString\\today{%
3304     <\language name date>[convert]%
3305     {\the\year}{\the\month}{\the\day}}}%
3306     \fi}%
3307     }}}

```

**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).

```

3308 \let\bbl@calendar@empty
3309 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3310   \@nameuse{bbl@ca#2}#1\@}
3311 \newcommand\babelDateSpace{\nobreakspace}
3312 \newcommand\babelDateDot{.\@} % TODO. \let instead of repeating
3313 \newcommand\babelDated[1]{\number#1}
3314 \newcommand\babelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3315 \newcommand\babelDateM[1]{\number#1}
3316 \newcommand\babelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3317 \newcommand\babelDateMMMM[1]{%
3318   \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3319 \newcommand\babelDatey[1]{\number#1}%
3320 \newcommand\babelDateyy[1]{%
3321   \ifnum#1<10 0\number#1 %
3322   \else\ifnum#1<100 \number#1 %
3323   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3324   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3325   \else
3326     \bbl@error
3327     {Currently two-digit years are restricted to the\
3328     range 0-9999.}%
3329     {There is little you can do. Sorry.}%
3330     \fi\fi\fi\fi}}
3331 \newcommand\babelDateyyyy[1]{\number#1} % TODO - add leading 0
3332 \def\bbl@replace@finish@iii#1{%
3333   \bbl@exp{\def\\#1####1####2####3{\the\toks@}}
3334 \def\bbl@TG@date{%
3335   \bbl@replace\bbl@toreplace{[ ]}{\babelDateSpace}}%
3336   \bbl@replace\bbl@toreplace{[. ]}{\babelDateDot}}%
3337   \bbl@replace\bbl@toreplace{[d]}{\babelDated{####3}}%
3338   \bbl@replace\bbl@toreplace{[dd]}{\babelDatedd{####3}}%
3339   \bbl@replace\bbl@toreplace{[M]}{\babelDateM{####2}}%
3340   \bbl@replace\bbl@toreplace{[MM]}{\babelDateMM{####2}}%
3341   \bbl@replace\bbl@toreplace{[MMMM]}{\babelDateMMMM{####2}}%
3342   \bbl@replace\bbl@toreplace{[y]}{\babelDatey{####1}}%
3343   \bbl@replace\bbl@toreplace{[yy]}{\babelDateyy{####1}}%
3344   \bbl@replace\bbl@toreplace{[yyyy]}{\babelDateyyyy{####1}}%

```

```

3345 \bbl@replace\bbl@toreplace{[y|]{\bbl@datecctr[####1|}%
3346 \bbl@replace\bbl@toreplace{[m|]{\bbl@datecctr[####2|}%
3347 \bbl@replace\bbl@toreplace{[d|]{\bbl@datecctr[####3|}%
3348 \bbl@replace@finish@iii\bbl@toreplace}
3349 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3350 \def\bbl@xdatecctr[#1|#2]{\localenumeral{#2}{#1}}

```

#### Transforms.

```

3351 \let\bbl@release@transforms\@empty
3352 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3353 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3354 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3355   #1[#2]{#3}{#4}{#5}}
3356 \begingroup % A hack. TODO. Don't require an specific order
3357   \catcode`\%=12
3358   \catcode`\&=14
3359   \gdef\bbl@transforms#1#2#3{%&
3360     \directlua{
3361       local str = [=[#2]=]
3362       str = str:gsub('%.%d+%.%d+$', '')
3363       tex.print([[def\string\babeltempa{]] .. str .. [[]]])
3364     }&
3365     \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&
3366     \ifin@
3367       \in@{.0$}{#2$}&
3368     \ifin@
3369       \directlua{%& (\attribute) syntax
3370         local str = string.match([[ \bbl@KVP@transforms]],
3371           '%([^(|-)%([^\%])-\babeltempa')
3372         if str == nil then
3373           tex.print([[def\string\babeltempb{]])
3374         else
3375           tex.print([[def\string\babeltempb{,attribute=]] .. str .. [[]]])
3376         end
3377       }
3378       \toks@{#3}&
3379       \bbl@exp{%&
3380         \g@addto@macro\ \bbl@release@transforms{%&
3381           \relax & Closes previous \bbl@transforms@aux
3382           \ \bbl@transforms@aux
3383           \ #1{label=\babeltempa\babeltempb}{\language\the\toks@}}&
3384       \else
3385         \g@addto@macro\ \bbl@release@transforms{, {#3}}&
3386       \fi
3387     \fi}
3388 \endgroup

```

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

```

3389 \def\bbl@provide@lsys#1{%
3390   \bbl@ifunset{\bbl@lname@#1}%
3391     {\bbl@load@info{#1}}%
3392   {}%
3393   \bbl@csarg\let{lsys@#1}\@empty
3394   \bbl@ifunset{\bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3395   \bbl@ifunset{\bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3396   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3397   \bbl@ifunset{\bbl@lname@#1}{}%
3398   {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3399   \ifcase\bbl@engine\or\or
3400     \bbl@ifunset{\bbl@prehc@#1}{}%
3401     {\bbl@exp{\ \bbl@ifblank{\bbl@cs{prehc@#1}}}%
3402     }%
3403     {\ifx\bbl@xenohyph\undefined

```

```

3404 \global\let\bbl@xenoxyph\bbl@xenoxyph@d
3405 \ifx\AtBeginDocument\@notprerr
3406 \expandafter\@secondoftwo % to execute right now
3407 \fi
3408 \AtBeginDocument{%
3409 \bbl@patchfont{\bbl@xenoxyph}%
3410 \expandafter\selectlanguage\expandafter{\language}%
3411 \fi}%
3412 \fi
3413 \bbl@csarg\bbl@tglobal{lsys@#1}}
3414 \def\bbl@xenoxyph@d{%
3415 \bbl@ifset{\bbl@prehc{\language}}%
3416 {\ifnum\hyphenchar\font=\defaultshyphenchar
3417 \iffontchar\font\bbl@cl{prehc}\relax
3418 \hyphenchar\font\bbl@cl{prehc}\relax
3419 \else\iffontchar\font"200B
3420 \hyphenchar\font"200B
3421 \else
3422 \bbl@warning
3423 {Neither 0 nor ZERO WIDTH SPACE are available\\%
3424 in the current font, and therefore the hyphen\\%
3425 will be printed. Try changing the fontspec's\\%
3426 'HyphenChar' to another value, but be aware\\%
3427 this setting is not safe (see the manual).\\%
3428 Reported}%
3429 \hyphenchar\font\defaultshyphenchar
3430 \fi\fi
3431 \fi}%
3432 {\hyphenchar\font\defaultshyphenchar}}
3433 % \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).

```

3434 \def\bbl@load@info#1{%
3435 \def\BabelBeforeIni##1##2{%
3436 \begingroup
3437 \bbl@read@ini{##1}0%
3438 \endinput % babel- .tex may contain only preamble's
3439 \endgroup}% boxed, to avoid extra spaces:
3440 {\bbl@input@texini{##1}}}

```

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

```

3441 \def\bbl@setdigits#1#2#3#4#5{%
3442 \bbl@exp{%
3443 \def<\language digits>####1{% ie, \langdigits
3444 \<\bbl@digits@language>####1\\@nil}%
3445 \let\<\bbl@cnt@digits@language>\<\language digits>%
3446 \def<\language counter>####1{% ie, \langcounter
3447 \\\expandafter\<\bbl@counter@language>%
3448 \\\csname c####1\endcsname}%
3449 \def\<\bbl@counter@language>####1{% ie, \bbl@counter@lang
3450 \\\expandafter\<\bbl@digits@language>%
3451 \\\number####1\\@nil}}%
3452 \def\bbl@tempa##1##2##3##4##5{%
3453 \bbl@exp{% Wow, quite a lot of hashes! :-(
3454 \def\<\bbl@digits@language>#####1{%
3455 \\\ifx#####1\\@nil % ie, \bbl@digits@lang
3456 \\\else
3457 \\\ifx0#####1#1%
3458 \\\else\\\ifx1#####1#2%

```

[illegible]

```

3472 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={%
3473   \ifx\\#1%           % \\ before, in case #1 is multiletter
3474     \bbl@exp{%
3475       \def\\bbl@tempa####1{%
3476         \<ifcase>####1\space\the\toks@\<else>\\@ctrerrr\<fi>}}%
3477   \else
3478     \toks@\expandafter{\the\toks@\or #1}%
3479     \expandafter\bbl@buildifcase
3480 \fi}

```

```

3481 \newcommand\localenatural[2]{\bbl@cs{cntr@#1\language}\#2}%
3482 \def\bbl@localecntr#1#2{\localenatural{#2}{#1}}
3483 \newcommand\localecounter[2]{%
3484   \expandafter\bbl@localecntr
3485   \expandafter{\number\csname c@#2\endcsname}{#1}}
3486 \def\bbl@alphnumerical#1#2{%
3487   \expandafter\bbl@alphnumerical@i\number#2 76543210\@@{#1}}
3488 \def\bbl@alphnumerical@i#1#2#3#4#5#6#7#8\@@#9{%
3489   \ifcase\car#8\@nil\or    % Currenty <10000, but prepared for bigger
3490     \bbl@alphnumerical@ii{#9}000000#1\or
3491     \bbl@alphnumerical@ii{#9}00000#1#2\or
3492     \bbl@alphnumerical@ii{#9}0000#1#2#3\or
3493     \bbl@alphnumerical@ii{#9}000#1#2#3#4\else
3494     \bbl@alphnum@invalid{>9999}%
3495   \fi}
3496 \def\bbl@alphnumerical@ii#1#2#3#4#5#6#7#8{%
3497   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8\language}%
3498     {\bbl@cs{cntr@#1.4\language}\#5%
3499     \bbl@cs{cntr@#1.3\language}\#6%
3500     \bbl@cs{cntr@#1.2\language}\#7%
3501     \bbl@cs{cntr@#1.1\language}\#8%
3502     \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3503       \bbl@ifunset{bbl@cntr@#1.S.321\language}{}%
3504       {\bbl@cs{cntr@#1.S.321\language}}%
3505     \fi}%
3506   {\bbl@cs{cntr@#1.F.\number#5#6#7#8\language}}
3507 \def\bbl@alphnum@invalid#1{%
3508   \bbl@error{Alphabetic numeral too large (#1)}%
3509   {Currently this is the limit.}}

```

```
3510 \def\bbl@localeinfo#1#2{%
3511   \bbl@ifunset{bbl@info@#2}{#1}%
```

```

3512     {\bbl@ifunset{bbl@csname bbl@info@#2\endcsname @\languagename}{#1}%
3513     {\bbl@cs{csname bbl@info@#2\endcsname @\languagename}}}%
3514 \newcommand\localeinfo[1]{%
3515   \ifx*#1@empty % TODO. A bit hackish to make it expandable.
3516     \bbl@afterelse\bbl@localeinfo}%
3517   \else
3518     \bbl@localeinfo
3519     {\bbl@error{I've found no info for the current locale.\\%
3520               The corresponding ini file has not been loaded\\%
3521               Perhaps it doesn't exist}%
3522     {See the manual for details.}}%
3523   {#1}%
3524 \fi}
3525 % \@namedef{bbl@info@name.locale}{lname}
3526 \@namedef{bbl@info@tag.ini}{lini}
3527 \@namedef{bbl@info@name.english}{elname}
3528 \@namedef{bbl@info@name.opentype}{lname}
3529 \@namedef{bbl@info@tag.bcp47}{tbcpl}
3530 \@namedef{bbl@info@language.tag.bcp47}{lbcp}
3531 \@namedef{bbl@info@tag.opentype}{lotf}
3532 \@namedef{bbl@info@script.name}{esname}
3533 \@namedef{bbl@info@script.name.opentype}{sname}
3534 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3535 \@namedef{bbl@info@script.tag.opentype}{sotf}
3536 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3537 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3538 % Extensions are dealt with in a special way
3539 % Now, an internal \LaTeX{} macro:
3540 \providecommand\BCPdata[1]{\localeinfo*{#1.tag.bcp47}}

```

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

```

3541 <(*More package options)> \equiv
3542 \DeclareOption{ensureinfo=off}{}
3543 <(/More package options)>
3544 %
3545 \let\bbl@ensureinfo\gobble
3546 \newcommand\BabelEnsureInfo{%
3547   \ifx\InputIfFileExists\undefined\else
3548     \def\bbl@ensureinfo##1{%
3549       \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3550   \fi
3551   \bbl@foreach\bbl@loaded{%
3552     \def\languagename{##1}%
3553     \bbl@ensureinfo{##1}}}%
3554 \ifpackagewith{babel}{ensureinfo=off}{}%
3555 {\AtEndOfPackage{% Test for plain.
3556   \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.

```

3557 \newcommand\getlocaleproperty{%
3558   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3559 \def\bbl@getproperty@s#1#2#3{%
3560   \let#1\relax
3561   \def\bbl@elt##1##2##3{%
3562     \bbl@ifsamestring{##1/##2}{#3}%
3563     {\providecommand#1{##3}%
3564     \def\bbl@elt####1####2####3{}}}%
3565   {}}%
3566   \bbl@cs{inidata@#2}}%
3567 \def\bbl@getproperty@x#1#2#3{%
3568   \bbl@getproperty@s{#1}{#2}{#3}%
3569   \ifx#1\relax

```

```

3570 \bbl@error
3571 {Unknown key for locale '#2':\%
3572 #3\}%
3573 \string#1 will be set to \relax}%
3574 {Perhaps you misspelled it.}%
3575 \fi}
3576 \let\bbl@ini@loaded\@empty
3577 \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.

```

3578 \newcommand\babeladjust[1]{% TODO. Error handling.
3579 \bbl@forkv{#1}{%
3580 \bbl@ifunset{bbl@ADJ@##1@##2}%
3581 {\bbl@cs{ADJ@##1}{##2}}%
3582 {\bbl@cs{ADJ@##1@##2}}}
3583 %
3584 \def\bbl@adjust@lua#1#2{%
3585 \ifvmode
3586 \ifnum\currentgrouplevel=\z@
3587 \directlua{ Babel.#2 }%
3588 \expandafter\expandafter\expandafter\@gobble
3589 \fi
3590 \fi
3591 {\bbl@error % The error is gobbled if everything went ok.
3592 {Currently, #1 related features can be adjusted only\%
3593 in the main vertical list.}%
3594 {Maybe things change in the future, but this is what it is.}}}
3595 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3596 \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3597 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3598 \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3599 \@namedef{bbl@ADJ@bidi.text@on}{%
3600 \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3601 \@namedef{bbl@ADJ@bidi.text@off}{%
3602 \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3603 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3604 \bbl@adjust@lua{bidi}{digits_mapped=true}}
3605 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3606 \bbl@adjust@lua{bidi}{digits_mapped=false}}
3607 %
3608 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3609 \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3610 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3611 \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3612 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3613 \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3614 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3615 \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3616 \@namedef{bbl@ADJ@justify.arabic@on}{%
3617 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3618 \@namedef{bbl@ADJ@justify.arabic@off}{%
3619 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3620 %
3621 \def\bbl@adjust@layout#1{%
3622 \ifvmode
3623 #1%
3624 \expandafter\@gobble
3625 \fi
3626 {\bbl@error % The error is gobbled if everything went ok.
3627 {Currently, layout related features can be adjusted only\%

```



```

3628     in vertical mode.}%
3629     {Maybe things change in the future, but this is what it is.}}
3630 \@namedef{bbl@ADJ@layout.tabular@on}{%
3631     \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}}
3632 \@namedef{bbl@ADJ@layout.tabular@off}{%
3633     \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}}
3634 \@namedef{bbl@ADJ@layout.lists@on}{%
3635     \bbl@adjust@layout{\let\list\bbl@NL@list}}
3636 \@namedef{bbl@ADJ@layout.lists@off}{%
3637     \bbl@adjust@layout{\let\list\bbl@OL@list}}
3638 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
3639     \bbl@activateposthyphen}
3640 %
3641 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3642     \bbl@bcpallowedtrue}
3643 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3644     \bbl@bcpallowedfalse}
3645 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3646     \def\bbl@bcp@prefix{#1}}
3647 \def\bbl@bcp@prefix{bcp47-}
3648 \@namedef{bbl@ADJ@autoload.options}#1{%
3649     \def\bbl@autoload@options{#1}}
3650 \let\bbl@autoload@bcptoptions\@empty
3651 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3652     \def\bbl@autoload@bcptoptions{#1}}
3653 \newif\ifbbl@bcptoname
3654 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3655     \bbl@bcptonametrue}
3656 \BabelEnsureInfo{
3657 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3658     \bbl@bcptonamefalse}
3659 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3660     \directlua{ Babel.ignore_pre_char = function(node)
3661         return (node.lang == \the\csname l@nohyphenation\endcsname)
3662     end }}
3663 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3664     \directlua{ Babel.ignore_pre_char = function(node)
3665         return false
3666     end }}
3667 \@namedef{bbl@ADJ@select.write@shift}{%
3668     \let\bbl@restorelastskip\relax
3669     \def\bbl@savelastskip{%
3670         \let\bbl@restorelastskip\relax
3671         \ifvmode
3672             \ifdim\lastskip=\z@
3673                 \let\bbl@restorelastskip\nobreak
3674             \else
3675                 \bbl@exp{%
3676                     \def\\bbl@restorelastskip{%
3677                         \skip@=\the\lastskip
3678                         \\nobreak \vskip-\skip@ \vskip\skip@}}%
3679                 \fi
3680             \fi}}
3681 \@namedef{bbl@ADJ@select.write@keep}{%
3682     \let\bbl@restorelastskip\relax
3683     \let\bbl@savelastskip\relax}
3684 \@namedef{bbl@ADJ@select.write@omit}{%
3685     \let\bbl@restorelastskip\relax
3686     \def\bbl@savelastskip##1\bbl@restorelastskip{}}

```

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

```

3687 \ifx\directlua\@undefined\else
3688     \ifx\bbl@luapatterns\@undefined

```

```

3689 \input luababel.def
3690 \fi
3691 \fi

```

Continue with  $\LaTeX$ .

```

3692 </package | core>
3693 <*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.

```

3694 <(*More package options)> ≡
3695 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3696 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3697 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3698 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3699 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3700 <\/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.

```

3701 \bbl@trace{Cross referencing macros}
3702 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3703   \def\@newl@bel#1#2#3{%
3704     {\@safe@activestrue
3705       \bbl@ifunset{#1@#2}%
3706       \relax
3707       {\gdef\@multiplelabels{%
3708         \@latex@warning@no@line{There were multiply-defined labels}}%
3709         \@latex@warning@no@line{Label `#2' multiply defined}}%
3710       \global\@namedef{#1@#2}{#3}}}%

```

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

```

3711 \CheckCommand*\@testdef[3]{%
3712   \def\reserved@a{#3}%
3713   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3714   \else
3715     \@tempswatrue
3716   \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.

```

3717 \def\@testdef#1#2#3{% TODO. With @samestring?
3718   \@safe@activestrue
3719   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3720   \def\bbl@tempb{#3}%
3721   \@safe@activesfalse
3722   \ifx\bbl@tempa\relax
3723     \else

```

```

3724 \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3725 \fi
3726 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3727 \ifx\bbl@tempa\bbl@tempb
3728 \else
3729 \@tempswatrue
3730 \fi}
3731 \fi

```

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

```

3732 \bbl@xin@{R}\bbl@opt@safe
3733 \ifin@
3734 \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3735 \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3736 {\expandafter\strip@prefix\meaning\ref}%
3737 \ifin@
3738 \bbl@redefine\@kernel@ref#1{%
3739 \@safe@activetrue\org@@kernel@ref{#1}\@safe@activetruefalse}
3740 \bbl@redefine\@kernel@pageref#1{%
3741 \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activetruefalse}
3742 \bbl@redefine\@kernel@sref#1{%
3743 \@safe@activetrue\org@@kernel@sref{#1}\@safe@activetruefalse}
3744 \bbl@redefine\@kernel@spageref#1{%
3745 \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activetruefalse}
3746 \else
3747 \bbl@redefineroobust\ref#1{%
3748 \@safe@activetrue\org@ref{#1}\@safe@activetruefalse}
3749 \bbl@redefineroobust\pageref#1{%
3750 \@safe@activetrue\org@pageref{#1}\@safe@activetruefalse}
3751 \fi
3752 \else
3753 \let\org@ref\ref
3754 \let\org@pageref\pageref
3755 \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.

```

3756 \bbl@xin@{B}\bbl@opt@safe
3757 \ifin@
3758 \bbl@redefine\@citex[#1]#2{%
3759 \@safe@activetrue\edef\@tempa{#2}\@safe@activetruefalse
3760 \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.

```

3761 \AtBeginDocument{%
3762 \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.)

```

3763 \def\@citex[#1][#2]#3{%
3764 \@safe@activetrue\edef\@tempa{#3}\@safe@activetruefalse
3765 \org@@citex[#1][#2]{\@tempa}}%
3766 }{}

```

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

```

3767 \AtBeginDocument{%
3768   \ifpackageloaded{cite}{%
3769     \def\citex[#1]#2{%
3770       \safe@activetrue\org@citex[#1]#2}\safe@activesfalse}%
3771     }{}}

```

\nocite The macro \nocite which is used to instruct BiB<sub>T</sub><sub>E</sub>X to extract uncited references from the database.

```

3772 \bbl@redefine\nocite#1{%
3773   \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 \bbl@cite@choice in such a way that it redefines itself with the proper definition. We call \bbl@cite@choice to select the proper definition for \bbl@cite. This new definition is then activated.

```

3774 \bbl@redefine\bibcite{%
3775   \bbl@cite@choice
3776   \bibcite}

```

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

```

3777 \def\bbl@bibcite#1#2{%
3778   \org@bibcite{#1}{\safe@activesfalse#2}}

```

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

```

3779 \def\bbl@cite@choice{%
3780   \global\let\bbl@cite\bbl@bibcite
3781   \ifpackageloaded{natbib}{\global\let\bbl@cite\org@bibcite}}%
3782   \ifpackageloaded{cite}{\global\let\bbl@cite\org@bibcite}}%
3783   \global\let\bbl@cite@choice\relax}

```

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

```

3784 \AtBeginDocument{\bbl@cite@choice}

```

\@bibitem One of the two internal <sub>T</sub><sub>E</sub>X macros called by \bibitem that write the citation label on the .aux file.

```

3785 \bbl@redefine\@bibitem#1{%
3786   \safe@activetrue\org@bibitem{#1}\safe@activesfalse}
3787 \else
3788   \let\org@nocite\nocite
3789   \let\org@@citex\citex
3790   \let\org@bibcite\bibcite
3791   \let\org@@bibitem\bibitem
3792 \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.

```

3793 \bbl@trace{Marks}
3794 \IfBabelLayout{sectioning}
3795   {\ifx\bbl@opt@headfoot\@nnil

```

```

3796 \g@addto@macro\@resetactivechars{%
3797 \set@typeset@protect
3798 \expandafter\select@language@x\expandafter{\bbl@main@language}%
3799 \let\protect\noexpand
3800 \ifcase\bbl@bidimode\else % Only with bidi. See also above
3801 \edef\thepage{%
3802 \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3803 \fi}%
3804 \fi}
3805 {\ifbbl@single\else
3806 \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
3807 \markright#1{%
3808 \bbl@ifblank{#1}%
3809 {\org@markright{}}%
3810 {\toks@{#1}%
3811 \bbl@exp{%
3812 \\\org@markright{\protect\\foreignlanguage{\language}%
3813 {\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.)

```

3814 \ifx\@mkboth\markboth
3815 \def\bbl@tempc{\let\@mkboth\markboth}
3816 \else
3817 \def\bbl@tempc{}
3818 \fi
3819 \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
3820 \markboth#1#2{%
3821 \protected@edef\bbl@tempb##1{%
3822 \protect\foreignlanguage
3823 {\language}\protect\bbl@restore@actives##1}%
3824 \bbl@ifblank{#1}%
3825 {\toks@{}}%
3826 {\toks@\expandafter{\bbl@tempb{#1}}}%
3827 \bbl@ifblank{#2}%
3828 {\@temptokena{}}%
3829 {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3830 \bbl@exp{\org@markboth{\the\toks@}{\the\@temptokena}}
3831 \bbl@tempc
3832 \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.

```

3833 \bbl@trace{Preventing clashes with other packages}
3834 \ifx\org@ref\@undefined\else
3835   \bbl@xin@{R}\bbl@opt@safe
3836   \ifin@
3837     \AtBeginDocument{%
3838       \@ifpackageloaded{ifthen}{%
3839         \bbl@redefine@long\ifthenelse#1#2#3{%
3840           \let\bbl@temp@pref\pageref
3841           \let\pageref\org@pageref
3842           \let\bbl@temp@ref\ref
3843           \let\ref\org@ref
3844           \@safe@activestrue
3845           \org@ifthenelse{#1}%
3846             {\let\pageref\bbl@temp@pref
3847              \let\ref\bbl@temp@ref
3848              \@safe@activesfalse
3849              #2}%
3850             {\let\pageref\bbl@temp@pref
3851              \let\ref\bbl@temp@ref
3852              \@safe@activesfalse
3853              #3}%
3854           }%
3855         }{}%
3856       }
3857 \fi

```

### 8.3.2 varioref

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

```

3858 \AtBeginDocument{%
3859   \@ifpackageloaded{varioref}{%
3860     \bbl@redefine\@vpageref#1[#2]#3{%
3861       \@safe@activestrue
3862       \org@@@vpageref{#1}#2#3}%
3863     \@safe@activesfalse}%
3864   \bbl@redefine\vrefpagemum#1#2{%
3865     \@safe@activestrue
3866     \org@vrefpagemum{#1}#2}%
3867   \@safe@activesfalse}%

```

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

```

3868   \expandafter\def\csname Ref \endcsname#1{%
3869     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3870   }{}%
3871 }
3872 \fi

```

### 8.3.3 hhlne

`\vhline` Delaying the activation of the shorthand characters has introduced a problem with the `hhlne` 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.

```

3873 \AtEndOfPackage{%
3874   \AtBeginDocument{%
3875     \ifpackageloaded{hhline}%
3876       {\expandafter\ifx\csname normal@char\string\endcsname\relax
3877         \else
3878           \makeatletter
3879           \def\@currname{hhline}\input{hhline.sty}\makeatother
3880           \fi}%
3881     {}}

```

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

```

3882 \def\substitutefontfamily#1#2#3{%
3883   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3884   \immediate\write15{%
3885     \string\ProvidesFile{#1#2.fd}%
3886     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3887     \space generated font description file]^J
3888     \string\DeclareFontFamily{#1}{#2}{}}^J
3889     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}}^J
3890     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}}^J
3891     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}}^J
3892     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}}^J
3893     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}}^J
3894     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}}^J
3895     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}}^J
3896     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}}^J
3897   }%
3898   \closeout15
3899 }
3900 \@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`

```

3901 \bbl@trace{Encoding and fonts}
3902 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3903 \newcommand\BabelNonText{TS1,T3,TS3}
3904 \let\org@TeX\TeX
3905 \let\org@LaTeX\LaTeX
3906 \let\ensureascii\@firstofone
3907 \AtBeginDocument{%
3908   \def\@elt#1{, #1,}%
3909   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3910   \let\@elt\relax
3911   \let\bbl@tempb\@empty
3912   \def\bbl@tempc{OT1}%
3913   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3914     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3915   \bbl@foreach\bbl@tempa{%
3916     \bbl@xin@{#1}{\BabelNonASCII}%
3917     \ifin@
3918       \def\bbl@tempb{#1}% Store last non-ascii
3919     \else\bbl@xin@{#1}{\BabelNonText}% Pass
3920     \ifin@\else
3921       \def\bbl@tempc{#1}% Store last ascii

```

```

3922     \fi
3923     \fi}%
3924     \ifx\bbbl@tempb\@empty\else
3925         \bbbl@xin@{\,cf@encoding,}{\,BabelNonASCII,\BabelNonText,}%
3926         \ifin@\else
3927             \edef\bbbl@tempc{\cf@encoding}% The default if ascii wins
3928             \fi
3929             \edef\ensureascii#1{%
3930                 {\noexpand\fontencoding{\bbbl@tempc}\noexpand\selectfont#1}}%
3931             \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3932             \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3933             \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.

```

3934 \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.

```

3935 \AtBeginDocument{%
3936     \@ifpackageloaded{fontspec}%
3937     {\xdef\latinencoding{%
3938         \ifx\UTFencname\@undefined
3939             EU\ifcase\bbbl@engine\or2\or1\fi
3940         \else
3941             \UTFencname
3942         \fi}}%
3943     {\gdef\latinencoding{OT1}%
3944         \ifx\cf@encoding\bbbl@t@one
3945             \xdef\latinencoding{\bbbl@t@one}%
3946         \else
3947             \def\@elt#1{, #1,}%
3948             \edef\bbbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3949             \let\@elt\relax
3950             \bbbl@xin@{, T1, }\bbbl@tempa
3951             \ifin@
3952                 \xdef\latinencoding{\bbbl@t@one}%
3953             \fi
3954             \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.

```

3955 \DeclareRobustCommand{\latintext}{%
3956     \fontencoding{\latinencoding}\selectfont
3957     \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.

```

3958 \ifx\@undefined\DeclareTextFontCommand
3959     \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3960 \else
3961     \DeclareTextFontCommand{\textlatin}{\latintext}
3962 \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).

```

3963 \def\bbbl@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  $\TeX$  grouping.
- `luatex` can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but bidi text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As `Lua $\TeX$ -ja` shows, vertical typesetting is possible, too.

```
3964 \bbl@trace{Loading basic (internal) bidi support}
3965 \ifodd\bbl@engine
3966 \else % TODO. Move to txtbabel
3967   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3968     \bbl@error
3969       {The bidi method 'basic' is available only in\\%
3970       luatex. I'll continue with 'bidi=default', so\\%
3971       expect wrong results}%
3972     {See the manual for further details.}%
3973     \let\bbl@beforeforeign\leavevmode
3974     \AtEndOfPackage{%
3975       \EnableBabelHook{babel-bidi}%
3976       \bbl@xebidipar}
3977   \fi\fi
3978   \def\bbl@loadxebidi#1{%
3979     \ifx\RTLfootnotetext\@undefined
3980       \AtEndOfPackage{%
3981         \EnableBabelHook{babel-bidi}%
3982         \bbl@loadfontspec % bidi needs fontspec
3983         \usepackage#1{bidi}}%
3984     \fi}
3985   \ifnum\bbl@bidimode>200
3986     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3987       \bbl@tentative{bidi=bidi}
3988       \bbl@loadxebidi{}
3989     \or
3990       \bbl@loadxebidi{[rldocument]}
3991     \or
3992       \bbl@loadxebidi{}
3993     \fi
3994   \fi
3995 \fi
3996 % TODO? Separate:
3997 \ifnum\bbl@bidimode=\@ne
3998   \let\bbl@beforeforeign\leavevmode
3999   \ifodd\bbl@engine
4000     \newattribute\bbl@attr@dir
4001     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
4002     \bbl@exp{\output{\bodydir\pagedir\the\output}}
4003   \fi
4004   \AtEndOfPackage{%
```

```

4005 \EnableBabelHook{babel-bidi}%
4006 \ifodd\bbbl@engine\else
4007 \bbbl@xebidipar
4008 \fi}
4009 \fi

```

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

```

4010 \bbbl@trace{Macros to switch the text direction}
4011 \def\bbbl@alscripts{,Arabic,Syriac,Thaana,}
4012 \def\bbbl@rscripts{% TODO. Base on codes ??
4013 ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
4014 Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
4015 Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
4016 Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
4017 Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
4018 Old South Arabian,}%
4019 \def\bbbl@provide@dirs#1{%
4020 \bbbl@xin@{\csname bbl@sname@#1\endcsname}{\bbbl@alscripts\bbbl@rscripts}%
4021 \ifin@
4022 \global\bbbl@csarg\chardef{wdir@#1}\@ne
4023 \bbbl@xin@{\csname bbl@sname@#1\endcsname}{\bbbl@alscripts}%
4024 \ifin@
4025 \global\bbbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
4026 \fi
4027 \else
4028 \global\bbbl@csarg\chardef{wdir@#1}\z@
4029 \fi
4030 \ifodd\bbbl@engine
4031 \bbbl@csarg\ifcase{wdir@#1}%
4032 \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
4033 \or
4034 \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
4035 \or
4036 \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
4037 \fi
4038 \fi}
4039 \def\bbbl@switchdir{%
4040 \bbbl@ifunset{bbl@lsys@\languagename}{\bbbl@provide@lsys{\languagename}}}%
4041 \bbbl@ifunset{bbl@wdir@\languagename}{\bbbl@provide@dirs{\languagename}}}%
4042 \bbbl@exp{\bbbl@setdirs\bbbl@cl{wdir}}}%
4043 \def\bbbl@setdirs#1{% TODO - math
4044 \ifcase\bbbl@select@type % TODO - strictly, not the right test
4045 \bbbl@bodydir{#1}%
4046 \bbbl@pardir{#1}%
4047 \fi
4048 \bbbl@textdir{#1}}
4049 % TODO. Only if \bbbl@bidimode > 0?:
4050 \AddBabelHook{babel-bidi}{afterextras}{\bbbl@switchdir}
4051 \DisableBabelHook{babel-bidi}

```

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

```

4052 \ifodd\bbbl@engine % luatex=1
4053 \else % pdftex=0, xetex=2
4054 \newcount\bbbl@dirlevel
4055 \chardef\bbbl@thetextdir\z@
4056 \chardef\bbbl@thepardir\z@
4057 \def\bbbl@textdir#1{%
4058 \ifcase#1\relax
4059 \chardef\bbbl@thetextdir\z@
4060 \bbbl@textdir@i\beginL\endL
4061 \else
4062 \chardef\bbbl@thetextdir\@ne
4063 \bbbl@textdir@i\beginR\endR

```

```

4064 \fi}
4065 \def\bbl@textdir@i#1#2{%
4066 \ifhmode
4067 \ifnum\currentgrouplevel>\z@
4068 \ifnum\currentgrouplevel=\bbl@dirlevel
4069 \bbl@error{Multiple bidi settings inside a group}%
4070 {I'll insert a new group, but expect wrong results.}%
4071 \bgroup\aftergroup#2\aftergroup\egroup
4072 \else
4073 \ifcase\currentgrouptype\or % 0 bottom
4074 \aftergroup#2% 1 simple {}
4075 \or
4076 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4077 \or
4078 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4079 \or\or\or % vbox vtop align
4080 \or
4081 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4082 \or\or\or\or\or\or % output math disc insert vcent mathchoice
4083 \or
4084 \aftergroup#2% 14 \beginngroup
4085 \else
4086 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4087 \fi
4088 \fi
4089 \bbl@dirlevel\currentgrouplevel
4090 \fi
4091 #1%
4092 \fi}
4093 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4094 \let\bbl@bodydir\@gobble
4095 \let\bbl@pagedir\@gobble
4096 \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).

```

4097 \def\bbl@xebidipar{%
4098 \let\bbl@xebidipar\relax
4099 \TeXeTstate\@ne
4100 \def\bbl@xeverypar{%
4101 \ifcase\bbl@thepardir
4102 \ifcase\bbl@thetextdir\else\beginR\fi
4103 \else
4104 {\setbox\z@\lastbox\beginR\box\z@}%
4105 \fi}%
4106 \let\bbl@severypar\everypar
4107 \newtoks\everypar
4108 \everypar=\bbl@severypar
4109 \bbl@severypar{\bbl@xeverypar\the\everypar}}
4110 \ifnum\bbl@bidimode>200
4111 \let\bbl@textdir@i\@gobbletwo
4112 \let\bbl@xebidipar\@empty
4113 \AddBabelHook{bidi}{foreign}{%
4114 \def\bbl@tempa{\def\BabelText####1}%
4115 \ifcase\bbl@thetextdir
4116 \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4117 \else
4118 \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4119 \fi}
4120 \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4121 \fi
4122 \fi

```

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

```

4123 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4124 \AtBeginDocument{%
4125   \ifx\pdfstringdefDisableCommands\undefined\else
4126     \ifx\pdfstringdefDisableCommands\relax\else
4127       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4128     \fi
4129   \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`.

```

4130 \bbl@trace{Local Language Configuration}
4131 \ifx\loadlocalcfg\undefined
4132   \@ifpackagewith{babel}{noconfigs}%
4133   {\let\loadlocalcfg@gobble}%
4134   {\def\loadlocalcfg#1{%
4135     \InputIfFileExists{#1.cfg}%
4136     {\typeout{*****^J%
4137               * Local config file #1.cfg used^^J%
4138               *}}}%
4139   \@empty}}
4140 \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).

```

4141 \bbl@trace{Language options}
4142 \let\bbl@afterlang\relax
4143 \let\BabelModifiers\relax
4144 \let\bbl@loaded\@empty
4145 \def\bbl@load@language#1{%
4146   \InputIfFileExists{#1.ldf}%
4147   {\edef\bbl@loaded{\CurrentOption
4148     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4149     \expandafter\let\expandafter\bbl@afterlang
4150     \csname\CurrentOption.ldf-h@@k\endcsname
4151     \expandafter\let\expandafter\BabelModifiers
4152     \csname\bbl@mod@\CurrentOption\endcsname}%
4153   {\bbl@error{%
4154     Unknown option '\CurrentOption'. Either you misspelled it\\%
4155     or the language definition file \CurrentOption.ldf was not found}}%
4156     Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4157     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4158     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.

```

4159 \def\bbl@try@load@lang#1#2#3{%
4160   \IfFileExists{\CurrentOption.ldf}%
4161   {\bbl@load@language{\CurrentOption}}%
4162   {#1\bbl@load@language{#2}#3}}
4163 %
4164 \DeclareOption{hebrew}{%
4165   \input{rlbabel.def}%

```

```

4166 \bbl@load@language{hebrew}}
4167 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4168 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4169 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4170 \DeclareOption{polutonikogreek}{%
4171   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4172 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4173 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4174 \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.

```

4175 \ifx\bbl@opt@config\@nnil
4176   \@ifpackagewith{babel}{noconfigs}{}%
4177   {\InputIfFileExists{bblopts.cfg}%
4178     {\typeout{*****^J%
4179               * Local config file bblopts.cfg used^J%
4180               *}}%
4181     {}}%
4182 \else
4183   \InputIfFileExists{\bbl@opt@config.cfg}%
4184   {\typeout{*****^J%
4185             * Local config file \bbl@opt@config.cfg used^J%
4186             *}}%
4187   {\bbl@error{%
4188     Local config file '\bbl@opt@config.cfg' not found}%
4189     Perhaps you misspelled it.}}%
4190 \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 `bbbl@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.

```

4191 \ifx\bbl@opt@main\@nnil
4192   \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4193     \let\bbl@tempb\@empty
4194     \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4195     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4196     \bbl@foreach\bbl@tempb{% \bbl@tempb is a reversed list
4197       \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4198         \ifodd\bbl@iniflag % =
4199           \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}}%
4200         \else % n +=
4201           \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}}%
4202       \fi
4203     \fi}%
4204 \fi
4205 \else
4206   \bbl@info{Main language set with 'main='. Except if you have\\%
4207     problems, prefer the default mechanism for setting\\%
4208     the main language. Reported}%
4209 \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`).

```

4210 \ifx\bbl@opt@main\@nnil\else
4211   \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
4212   \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4213 \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.

```

4214 \bbl@foreach\bbl@language@opts{%
4215   \def\bbl@tempa{#1}%
4216   \ifx\bbl@tempa\bbl@opt@main\else
4217     \ifnum\bbl@iniflag<\tw@    % 0 0 (other = ldf)
4218       \bbl@ifunset{ds@#1}%
4219       {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4220       {}%
4221     \else
4222       \DeclareOption{#1}{%
4223         \bbl@ldfinit
4224         \babelprovide[import]{#1}%
4225         \bbl@afterldf{}}%
4226     \fi
4227   \fi}
4228 \bbl@foreach\@classoptionslist{%
4229   \def\bbl@tempa{#1}%
4230   \ifx\bbl@tempa\bbl@opt@main\else
4231     \ifnum\bbl@iniflag<\tw@    % 0 0 (other = ldf)
4232     \bbl@ifunset{ds@#1}%
4233     {\IfFileExists{#1.ldf}%
4234      {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4235      {}}%
4236     {}%
4237   \else
4238     \IfFileExists{babel-#1.tex}%
4239     {\DeclareOption{#1}{%
4240       \bbl@ldfinit
4241       \babelprovide[import]{#1}%
4242       \bbl@afterldf{}}}%
4243     {}%
4244   \fi
4245 \fi}

```

And we are done, because all options for this pass has been declared. Those already processed in the first pass are just ignored.

The options have to be processed in the order in which the user specified them (but remember class options are processes before):

```

4246 \def\AfterBabelLanguage#1{%
4247   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang{}}
4248   \DeclareOption*{}
4249   \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.

```

4250 \bbl@trace{Option 'main'}
4251 \ifx\bbl@opt@main\@nnil
4252   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4253   \let\bbl@tempc\@empty
4254   \edef\bbl@templ{\bbl@loaded,}
4255   \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4256   \bbl@for\bbl@tempb\bbl@tempa{%
4257     \edef\bbl@tempd{\bbl@tempb,%}
4258     \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4259     \bbl@xin@{\bbl@tempd}{\bbl@templ}%
4260     \ifin\edef\bbl@tempc{\bbl@tempb}\fi}
4261   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4262   \expandafter\bbl@tempa\bbl@loaded,\@nnil

```

```

4263 \ifx\bbl@tempb\bbl@tempc\else
4264 \bbl@warning{%
4265     Last declared language option is '\bbl@tempc',\%
4266     but the last processed one was '\bbl@tempb'.\%
4267     The main language can't be set as both a global\%
4268     and a package option. Use 'main=\bbl@tempc' as\%
4269     option. Reported}
4270 \fi
4271 \else
4272 \ifodd\bbl@iniflag % case 1,3 (main is ini)
4273 \bbl@ldfinit
4274 \let\CurrentOption\bbl@opt@main
4275 \bbl@exp{% \bbl@opt@provide = empty if *
4276     \\\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4277 \bbl@afterldf{}
4278 \DeclareOption{\bbl@opt@main}{}
4279 \else % case 0,2 (main is ldf)
4280 \ifx\bbl@loadmain\relax
4281 \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4282 \else
4283 \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4284 \fi
4285 \ExecuteOptions{\bbl@opt@main}
4286 \@namedef{ds@\bbl@opt@main}{}%
4287 \fi
4288 \DeclareOption*{}
4289 \ProcessOptions*
4290 \fi
4291 \def\AfterBabelLanguage{%
4292 \bbl@error
4293 {Too late for \string\AfterBabelLanguage}%
4294 {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.

```

4295 \ifx\bbl@main@language\@undefined
4296 \bbl@info{%
4297     You haven't specified a language as a class or package\%
4298     option. I'll load 'nil'. Reported}
4299 \bbl@load@language{nil}
4300 \fi
4301 \</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  $\text{\TeX}$  users might want to use some of the features of the babel system too, care has to be taken that plain  $\text{\TeX}$  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  $\text{\TeX}$  and  $\text{\LaTeX}$ , some of it is for the  $\text{\LaTeX}$  case only.

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

A proxy file for `switch.def`

```

4302 \<*kernel>
4303 \let\bbl@onlyswitch\@empty
4304 \input babel.def
4305 \let\bbl@onlyswitch\@undefined
4306 \</kernel>
4307 \<*patterns>

```

## 10 Loading hyphenation patterns

The following code is meant to be read by  $\text{\texttt{iniT\TeX}}$  because it should instruct  $\text{\texttt{T\TeX}}$  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.

```
4308 <<Make sure ProvidesFile is defined>>
4309 \ProvidesFile{hyphen.cfg}[\<<date>> \<<version>> Babel hyphens]
4310 \xdef\bbl@format{\jobname}
4311 \def\bbl@version{\<<version>>}
4312 \def\bbl@date{\<<date>>}
4313 \ifx\AtBeginDocument\@undefined
4314   \def\@empty{}
4315 \fi
4316 <<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.

```
4317 \def\process@line#1#2 #3 #4 {%
4318   \ifx=#1%
4319     \process@synonym{#2}%
4320   \else
4321     \process@language{#1#2}{#3}{#4}%
4322   \fi
4323   \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.

```
4324 \toks@{}
4325 \def\bbl@languages{}
```

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

```
4326 \def\process@synonym#1{%
4327   \ifnum\last@language=\m@ne
4328     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4329   \else
4330     \expandafter\chardef\csname l@#1\endcsname\last@language
4331     \wlog{\string\l@#1=\string\language\the\last@language}%
4332     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4333       \csname\language\hyphenmins\endcsname
4334     \let\bbl@elt\relax
4335     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}}}%
4336   \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`.  $\text{\texttt{T\TeX}}$  does not keep track of these assignments. Therefore we try to detect such assignments and store them in the `\<lang>hyphenmins` macro. When no assignments were made we provide a default setting.



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

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

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

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

```

4337 \def\process@language#1#2#3{%
4338   \expandafter\addlanguage\csname l@#1\endcsname
4339   \expandafter\language\csname l@#1\endcsname
4340   \edef\language{#1}%
4341   \bbl@hook@everylanguage{#1}%
4342   % > luatex
4343   \bbl@get@enc#1::\@@@
4344   \begingroup
4345     \lefthyphenmin@m@ne
4346     \bbl@hook@loadpatterns{#2}%
4347     % > luatex
4348     \ifnum\lefthyphenmin=m@ne
4349       \else
4350         \expandafter\xdef\csname #1hyphenmins\endcsname{%
4351           \the\lefthyphenmin\the\righthyphenmin}%
4352       \fi
4353   \endgroup
4354   \def\bbl@tempa{#3}%
4355   \ifx\bbl@tempa\@empty\else
4356     \bbl@hook@loadexceptions{#3}%
4357     % > luatex
4358   \fi
4359   \let\bbl@elt\relax
4360   \edef\bbl@languages{%
4361     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4362   \ifnum\the\language=z@
4363     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4364       \set@hyphenmins\tw@\thr@@\relax
4365     \else
4366       \expandafter\expandafter\expandafter\set@hyphenmins
4367       \csname #1hyphenmins\endcsname
4368     \fi
4369     \the\toks@
4370     \toks@{}%
4371   \fi}

```

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

```

4372 \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.

```

4373 \def\bbl@hook@everylanguage#1{}
4374 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4375 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4376 \def\bbl@hook@loadkernel#1{%
4377   \def\addlanguage{\csname newlanguage\endcsname}%
4378   \def\adddialect##1##2{%

```

```

4379 \global\chardef##1##2\relax
4380 \wlog{\string##1 = a dialect from \string\language##2}}%
4381 \def\iflanguage##1{%
4382 \expandafter\ifx\csname l@##1\endcsname\relax
4383 \nolanerr{##1}%
4384 \else
4385 \ifnum\csname l@##1\endcsname=\language
4386 \expandafter\expandafter\expandafter\@firstoftwo
4387 \else
4388 \expandafter\expandafter\expandafter\@secondoftwo
4389 \fi
4390 \fi}%
4391 \def\providehyphenmins##1##2{%
4392 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4393 \namedef{##1hyphenmins}{##2}%
4394 \fi}%
4395 \def\set@hyphenmins##1##2{%
4396 \lefthyphenmin##1\relax
4397 \righthyphenmin##2\relax}%
4398 \def\selectlanguage{%
4399 \errhelp{Selecting a language requires a package supporting it}%
4400 \errmessage{Not loaded}}%
4401 \let\foreignlanguage\selectlanguage
4402 \let\otherlanguage\selectlanguage
4403 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4404 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4405 \def\setlocale{%
4406 \errhelp{Find an armchair, sit down and wait}%
4407 \errmessage{Not yet available}}%
4408 \let\uselocale\setlocale
4409 \let\locale\setlocale
4410 \let\selectlocale\setlocale
4411 \let\localename\setlocale
4412 \let\textlocale\setlocale
4413 \let\textlanguage\setlocale
4414 \let\languagetext\setlocale}
4415 \begingroup
4416 \def\AddBabelHook#1#2{%
4417 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4418 \def\next{\toks1}%
4419 \else
4420 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4421 \fi
4422 \next}
4423 \ifx\directlua\undefined
4424 \ifx\XeTeXinputencoding\undefined\else
4425 \input xebabel.def
4426 \fi
4427 \else
4428 \input luababel.def
4429 \fi
4430 \openin1 = babel-\bbl@format.cfg
4431 \ifeof1
4432 \else
4433 \input babel-\bbl@format.cfg\relax
4434 \fi
4435 \closein1
4436 \endgroup
4437 \bbl@hook@loadkernel{switch.def}

```

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

```
4438 \openin1 = language.dat
```

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

about this.

```
4439 \def\language{english}%
4440 \ifeof1
4441   \message{I couldn't find the file language.dat,\space
4442           I will try the file hyphen.tex}
4443   \input hyphen.tex\relax
4444   \chardef\l@english\z@
4445 \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`.

```
4446   \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.

```
4447   \loop
4448     \endlinechar\m@ne
4449     \read1 to \bbl@line
4450     \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.

```
4451   \if T\ifeof1F\fi T\relax
4452   \ifx\bbl@line\empty\else
4453     \edef\bbl@line{\bbl@line\space\space\space}%
4454     \expandafter\process@line\bbl@line\relax
4455   \fi
4456 \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.

```
4457 \begingroup
4458   \def\bbl@elt#1#2#3#4{%
4459     \global\language=#2\relax
4460     \gdef\language{#1}%
4461     \def\bbl@elt##1##2##3##4{}}%
4462   \bbl@languages
4463 \endgroup
4464 \fi
4465 \closein1
```

We add a message about the fact that babel is loaded in the format and with which language patterns to the `\everyjob` register.

```
4466 \if/\the\toks@\else
4467   \errhelp{language.dat loads no language, only synonyms}
4468   \errmessage{Orphan language synonym}
4469 \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.

```
4470 \let\bbl@line\@undefined
4471 \let\process@line\@undefined
4472 \let\process@synonym\@undefined
4473 \let\process@language\@undefined
4474 \let\bbl@get@enc\@undefined
4475 \let\bbl@hyph@enc\@undefined
4476 \let\bbl@tempa\@undefined
4477 \let\bbl@hook@loadkernel\@undefined
4478 \let\bbl@hook@everylanguage\@undefined
4479 \let\bbl@hook@loadpatterns\@undefined
```

```

4480 \let\bbl@hook@loadexceptions\@undefined
4481 \end{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].

```

4482 \langle *More package options \rangle \equiv
4483 \chardef\bbl@bidimode\z@
4484 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4485 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4486 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4487 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4488 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4489 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4490 \rangle /More package options \rangle

```

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.

```

4491 \langle *Font selection \rangle \equiv
4492 \bbl@trace{Font handling with fontspec}
4493 \ifx\ExplSyntaxOn\@undefined\else
4494   \def\bbl@fs@warn@nx#1#2{% \bbl@tempfs is the original macro
4495     \in@{, #1, }{, no-script, language-not-exist, }%
4496     \ifin\else\bbl@tempfs@nx{#1}{#2}\fi}
4497   \def\bbl@fs@warn@nxx#1#2#3{%
4498     \in@{, #1, }{, no-script, language-not-exist, }%
4499     \ifin\else\bbl@tempfs@nxx{#1}{#2}{#3}\fi}
4500   \def\bbl@loadfontspec{%
4501     \let\bbl@loadfontspec\relax
4502     \ifx\fontspec\@undefined
4503       \usepackage{fontspec}%
4504     \fi}%
4505 \fi
4506 \@onlypreamble\babelfont
4507 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4508   \bbl@foreach{#1}{%
4509     \expandafter\ifx\csname date##1\endcsname\relax
4510       \IfFileExists{babel-##1.tex}%
4511       {\babelprovide{##1}}%
4512     }%
4513   \fi}%
4514 \edef\bbl@tempa{#1}%
4515 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4516 \bbl@loadfontspec
4517 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4518 \bbl@bblfont}
4519 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4520   \bbl@ifunset{\bbl@tempb family}%
4521   {\bbl@providefam{\bbl@tempb}}%
4522   {}%
4523   % For the default font, just in case:
4524   \bbl@ifunset{\bbl@lsys\languagename}{\bbl@provide@lsys{\languagename}}{}%
4525   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4526   {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save \bbl@rmdflt@
4527   \bbl@exp{%
4528     \let\<\bbl@tempb dflt@\languagename>\<\bbl@tempb dflt@>%

```

```

4529      \bbl@font@set\<bbl@bbl@tempb dflt@language>%
4530      \<bbl@tempb default>\<bbl@tempb family>}}%
4531      {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4532      \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:

```

4533 \def\bbl@providefam#1{%
4534   \bbl@exp{%
4535     \\\newcommand\<#1default>{}% Just define it
4536     \\\bbl@add@list\\bbl@font@fams{#1}%
4537     \\\DeclareRobustCommand\<#1family>{%
4538       \\\not@math@alphabet\<#1family>\relax
4539       % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4540       \\\fontfamily\<#1default>%
4541       \<ifx>\\\UseHooks\\@undefined\<else>\\\UseHook{#1family}\<fi>%
4542       \\\selectfont}%
4543       \\\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.

```

4544 \def\bbl@nostdfont#1{%
4545   \bbl@ifunset{bbl@WFF@f@family}%
4546   {\bbl@csarg\gdef{WFF@f@family}}% Flag, to avoid dupl warns
4547   \bbl@infowarn{The current font is not a babel standard family:\%
4548     #1%
4549     \fontname\font\\%
4550     There is nothing intrinsically wrong with this warning, and\\%
4551     you can ignore it altogether if you do not need these\\%
4552     families. But if they are used in the document, you should be\\%
4553     aware 'babel' will not set Script and Language for them, so\\%
4554     you may consider defining a new family with \string\babelfont.\\%
4555     See the manual for further details about \string\babelfont.\\%
4556     Reported}}
4557   }%
4558 \gdef\bbl@switchfont{%
4559   \bbl@ifunset{bbl@lsys@language}{\bbl@provide@lsys{language}}}%
4560   \bbl@exp{% eg Arabic -> arabic
4561     \lowercase{\edef\bbl@tempa{\bbl@cl{sname}}}}%
4562   \bbl@foreach\bbl@font@fams{%
4563     \bbl@ifunset{bbl@##1dflt@language}% (1) language?
4564     {\bbl@ifunset{bbl@##1dflt@*bbl@tempa}% (2) from script?
4565     {\bbl@ifunset{bbl@##1dflt@}% 2=F - (3) from generic?
4566     {}% 123=F - nothing!
4567     {\bbl@exp{% 3=T - from generic
4568       \global\let\<bbl@##1dflt@language>%
4569       \<bbl@##1dflt@>}}}%
4570     {\bbl@exp{% 2=T - from script
4571       \global\let\<bbl@##1dflt@language>%
4572       \<bbl@##1dflt@*bbl@tempa>}}}%
4573     }% 1=T - language, already defined
4574   \def\bbl@tempa{\bbl@nostdfont}}%
4575   \bbl@foreach\bbl@font@fams{% don't gather with prev for
4576     \bbl@ifunset{bbl@##1dflt@language}%
4577     {\bbl@cs{famrst##1}%
4578     \global\bbl@csarg\let{famrst##1}\relax}%
4579     {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4580       \\\bbl@add\\originalTeX%
4581       \\\bbl@font@rst{\bbl@cl{##1dflt}}%
4582       \<##1default>\<##1family>{##1}}%
4583       \\\bbl@font@set\<bbl@##1dflt@language>% the main part!
4584       \<##1default>\<##1family>}}}%
4585   \bbl@ifrestoring{\bbl@tempa}%

```

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

```

4586 \ifx\fbfamily\@undefined\else % if latex
4587 \ifcase\bbl@engine % if pdftex
4588 \let\bbl@ckeckstdfonts\relax
4589 \else
4590 \def\bbl@ckeckstdfonts{%
4591 \begingroup
4592 \global\let\bbl@ckeckstdfonts\relax
4593 \let\bbl@tempa\@empty
4594 \bbl@foreach\bbl@font@fams{%
4595 \bbl@ifunset\bbl@##1dflt@{%
4596 {\@nameuse{##1family}}%
4597 \bbl@csarg\gdef{WFF@fbfamily}}}% Flag
4598 \bbl@exp{\bbl@add\bbl@tempa{* \<##1family>= \fbfamily\\}%
4599 \space\space\fontname\font\\}}%
4600 \bbl@csarg\xdef{##1dflt@}{fbfamily}%
4601 \expandafter\xdef\csname ##1default\endcsname{fbfamily}}%
4602 {}}%
4603 \ifx\bbl@tempa\@empty\else
4604 \bbl@infowarn{The following font families will use the default\\%
4605 settings for all or some languages:\\%
4606 \bbl@tempa
4607 There is nothing intrinsically wrong with it, but\\%
4608 'babel' will no set Script and Language, which could\\%
4609 be relevant in some languages. If your document uses\\%
4610 these families, consider redefining them with \string\babelfont.\\%
4611 Reported}%
4612 \fi
4613 \endgroup}
4614 \fi
4615 \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.

```

4616 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4617 \bbl@xin@{<>}{#1}%
4618 \ifin@
4619 \bbl@exp{\bbl@fontspec@set\\#1\expandafter@gobbletwo#1\\#3}%
4620 \fi
4621 \bbl@exp{% 'Unprotected' macros return prev values
4622 \def\\#2{#1}% eg, \rmdefault{\bbl@rmdflt@lang}
4623 \\bbl@ifsamestring{#2}{fbfamily}%
4624 {\\#3%
4625 \\bbl@ifsamestring{f@series}{bfdefault}{\\bfseries}}}%
4626 \let\\bbl@tempa\relax}%
4627 {}}}
4628 % TODO - next should be global?, but even local does its job. I'm
4629 % still not sure -- must investigate:
4630 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4631 \let\bbl@tempa\bbl@mapselect
4632 \let\bbl@mapselect\relax
4633 \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4634 \let#4\empty % Make sure \renewfontfamily is valid
4635 \bbl@exp{%
4636 \let\\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
4637 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4638 {\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4639 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4640 {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4641 \let\\bbl@tempfs@nx<__fontspec_warning:nx>%

```

```

4642 \let\<__fontspec_warning:nx>\bbl@fs@warn@nx
4643 \let\bbl@tempfs@nxx\<__fontspec_warning:nxx>%
4644 \let\<__fontspec_warning:nxx>\bbl@fs@warn@nxx
4645 \renewfontfamily\#4%
4646 [\bbl@cl{lsys},#2]{#3}% ie \bbl@exp{..}{#3}
4647 \bbl@exp{%
4648 \let\<__fontspec_warning:nx>\bbl@tempfs@nx
4649 \let\<__fontspec_warning:nxx>\bbl@tempfs@nxx}%
4650 \begingroup
4651 #4%
4652 \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4653 \endgroup
4654 \let#4\bbl@temp@fam
4655 \bbl@exp{\let\<bbl@stripslash#4\space>\bbl@temp@pfam
4656 \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.

```

4657 \def\bbl@font@rst#1#2#3#4{%
4658 \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.

```

4659 \def\bbl@font@fams{rm,sf,tt}
4660 <\/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.

```

4661 <(*Footnote changes)> ≡
4662 \bbl@trace{Bidi footnotes}
4663 \ifnum\bbl@bidimode>\z@
4664 \def\bbl@footnote#1#2#3{%
4665 \ifnextchar[%
4666 {\bbl@footnote@o{#1}{#2}{#3}}%
4667 {\bbl@footnote@x{#1}{#2}{#3}}}
4668 \long\def\bbl@footnote@x#1#2#3#4{%
4669 \bgroup
4670 \select@language@x{\bbl@main@language}%
4671 \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4672 \egroup}
4673 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4674 \bgroup
4675 \select@language@x{\bbl@main@language}%
4676 \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4677 \egroup}
4678 \def\bbl@footnotetext#1#2#3{%
4679 \ifnextchar[%
4680 {\bbl@footnotetext@o{#1}{#2}{#3}}%
4681 {\bbl@footnotetext@x{#1}{#2}{#3}}}
4682 \long\def\bbl@footnotetext@x#1#2#3#4{%
4683 \bgroup
4684 \select@language@x{\bbl@main@language}%
4685 \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4686 \egroup}
4687 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4688 \bgroup
4689 \select@language@x{\bbl@main@language}%
4690 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4691 \egroup}

```

```

4692 \def\BabelFootnote#1#2#3#4{%
4693 \ifx\bb1@fn@footnote\undefined
4694 \let\bb1@fn@footnote\footnote
4695 \fi
4696 \ifx\bb1@fn@footnotetext\undefined
4697 \let\bb1@fn@footnotetext\footnotetext
4698 \fi
4699 \bb1@ifblank{#2}%
4700 {\def#1{\bb1@footnote{\@firstofone}{#3}{#4}}
4701 \@namedef{\bb1@stripslash#1text}%
4702 {\bb1@footnotetext{\@firstofone}{#3}{#4}}}%
4703 {\def#1{\bb1@exp{\bb1@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4704 \@namedef{\bb1@stripslash#1text}%
4705 {\bb1@exp{\bb1@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4706 \fi
4707 <</Footnote changes>>

```

Now, the code.

```

4708 <*xetex>
4709 \def\BabelStringsDefault{unicode}
4710 \let\xebbl@stop\relax
4711 \AddBabelHook{xetex}{encodedcommands}{%
4712 \def\bb1@tempa{#1}%
4713 \ifx\bb1@tempa\empty
4714 \XeTeXinputencoding"bytes"%
4715 \else
4716 \XeTeXinputencoding"#1"%
4717 \fi
4718 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4719 \AddBabelHook{xetex}{stopcommands}{%
4720 \xebbl@stop
4721 \let\xebbl@stop\relax}
4722 \def\bb1@intraspace#1 #2 #3\@{%
4723 \bb1@csarg\gdef{\xeisp@{language}}%
4724 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4725 \def\bb1@intrapenalty#1\@{%
4726 \bb1@csarg\gdef{\xeipn@{language}}%
4727 {\XeTeXlinebreakpenalty #1\relax}}
4728 \def\bb1@provide@intraspace{%
4729 \bb1@xin@{/s}{/\bb1@cl{lnbrk}}%
4730 \ifin@ \else \bb1@xin@{/c}{/\bb1@cl{lnbrk}}\fi
4731 \ifin@
4732 \bb1@ifunset{\bb1@intsp@{language}}{%
4733 {\expandafter\ifx\csname \bb1@intsp@{language}\endcsname\empty\else
4734 \ifx\bb1@KVP@intraspace\@nnil
4735 \bb1@exp{%
4736 \bb1@intraspace\bb1@cl{intsp}\@}%
4737 \fi
4738 \ifx\bb1@KVP@intrapenalty\@nnil
4739 \bb1@intrapenalty0\@
4740 \fi
4741 \fi
4742 \ifx\bb1@KVP@intraspace\@nnil\else % We may override the ini
4743 \expandafter\bb1@intraspace\bb1@KVP@intraspace\@
4744 \fi
4745 \ifx\bb1@KVP@intrapenalty\@nnil\else
4746 \expandafter\bb1@intrapenalty\bb1@KVP@intrapenalty\@
4747 \fi
4748 \bb1@exp{%
4749 % TODO. Execute only once (but redundant):
4750 \bb1@add\<extras\language>%
4751 \XeTeXlinebreaklocale "\bb1@cl{tbcpr}"%
4752 \bb1@xeisp@{language}%

```



```

4753      \<bbl@xeipn@<language>\language>}%
4754      \\\bbl@tglobal\<extras\language>%
4755      \\\bbl@add\<noextras\language>{%
4756      \XeTeXlinebreaklocale ""}%
4757      \\\bbl@tglobal\<noextras\language>}%
4758      \ifx\bbl@ispace\@undefined
4759      \gdef\bbl@ispace{\bbl@cl{xeisp}}%
4760      \ifx\AtBeginDocument\@notprerr
4761      \expandafter\@secondoftwo % to execute right now
4762      \fi
4763      \AtBeginDocument{\bbl@patchfont{\bbl@ispace}}%
4764      \fi}%
4765  \fi}
4766  \ifx\DisableBabelHook\@undefined\endinput\fi
4767  \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4768  \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfont}
4769  \DisableBabelHook{babel-fontspec}
4770  <<Font selection>>
4771  \input txtbabel.def
4772  </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 T<sub>E</sub>X 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.

```

4773 <texxet>
4774 \providecommand\bbl@provide@intraspace{}
4775 \bbl@trace{Redefinitions for bidi layout}
4776 \def\bbl@sspre@caption{%
4777   \bbl@exp{\everyhbox{\\\bbl@texdir\bbl@cs{wdir}\bbl@main@language}}}%
4778 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4779 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4780 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4781 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4782   \def\@hangfrom#1{%
4783     \setbox\@tempboxa\hbox{#1}%
4784     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4785     \noindent\box\@tempboxa}
4786   \def\raggedright{%
4787     \let\@centercr
4788     \bbl@startskip\z@skip
4789     \@rightskip\flushglue
4790     \bbl@endskip\@rightskip
4791     \parindent\z@
4792     \parfillskip\bbl@startskip}
4793   \def\raggedleft{%
4794     \let\@centercr
4795     \bbl@startskip\flushglue
4796     \bbl@endskip\z@skip
4797     \parindent\z@
4798     \parfillskip\bbl@endskip}
4799 \fi
4800 \IfBabelLayout{lists}
4801 {\bbl@replace\list
4802   {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4803   \def\bbl@listleftmargin{%
4804     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4805   \ifcase\bbl@engine

```

```

4806 \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4807 \def\p@enumiii{\p@enumii}\theenumii{}\%
4808 \fi
4809 \bbl@sreplace\@verbatim
4810 {\leftskip\@totalleftmargin}%
4811 {\bbl@startskip\textwidth
4812 \advance\bbl@startskip-\linewidth}%
4813 \bbl@sreplace\@verbatim
4814 {\rightskip\z@skip}%
4815 {\bbl@endskip\z@skip}}%
4816 {}
4817 \IfBabelLayout{contents}
4818 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4819 \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4820 {}
4821 \IfBabelLayout{columns}
4822 {\bbl@sreplace\@outputdblcol{\hb@xt\textwidth}{\bbl@outputbox}%
4823 \def\bbl@outputbox#1{%
4824 \hb@xt\textwidth{%
4825 \hspace\columnwidth
4826 \hfil
4827 {\normalcolor\vrule \@width\columnseprule}%
4828 \hfil
4829 \hb@xt\columnwidth{\box\@leftcolumn \hss}%
4830 \hspace-\textwidth
4831 \hb@xt\columnwidth{\box\@outputbox \hss}%
4832 \hspace\columnsep
4833 \hspace\columnwidth}}}%
4834 {}
4835 \langle Footnote changes \rangle
4836 \IfBabelLayout{footnotes}%
4837 {\BabelFootnote\footnote\languagename{}}{}%
4838 \BabelFootnote\localfootnote\languagename{}}{}%
4839 \BabelFootnote\mainfootnote{}}{}{}%
4840 {}

```

Implicitly reverses sectioning labels in `bidibasic`, because the full stop is not in contact with L numbers any more. I think there must be a better way.

```

4841 \IfBabelLayout{counters}%
4842 {\let\bbl@latinarabic=\@arabic
4843 \def\@arabic#1{\bbl@subl\@latinarabic#1}}%
4844 \let\bbl@asciroman=\@roman
4845 \def\@roman#1{\bbl@subl\@ensureascii\bbl@asciroman#1}}%
4846 \let\bbl@asciiRoman=\@Roman
4847 \def\@Roman#1{\bbl@subl\@ensureascii\bbl@asciiRoman#1}}{}%
4848 \end{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).

```

4849 <*luatex>
4850 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4851 \bbl@trace{Read language.dat}
4852 \ifx\bbl@readstream\@undefined
4853 \csname newread\endcsname\bbl@readstream
4854 \fi
4855 \begingroup
4856 \toks@{}
4857 \count@ \z@ % 0=start, 1=0th, 2=normal
4858 \def\bbl@process@line#1#2 #3 #4 {%
4859 \ifx=#1%
4860 \bbl@process@synonym{#2}%
4861 \else
4862 \bbl@process@language{#1#2}{#3}{#4}%
4863 \fi
4864 \ignorespaces}
4865 \def\bbl@manylang{%
4866 \ifnum\bbl@last>\@ne
4867 \bbl@info{Non-standard hyphenation setup}%
4868 \fi
4869 \let\bbl@manylang\relax}
4870 \def\bbl@process@language#1#2#3{%
4871 \ifcase\count@
4872 \@ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
4873 \or
4874 \count@\tw@
4875 \fi
4876 \ifnum\count@=\tw@
4877 \expandafter\addlanguage\csname l@#1\endcsname
4878 \language\allocationnumber
4879 \chardef\bbl@last\allocationnumber
4880 \bbl@manylang
4881 \let\bbl@elt\relax
4882 \xdef\bbl@languages{%
4883 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4884 \fi
4885 \the\toks@
4886 \toks@{}}
4887 \def\bbl@process@synonym@aux#1#2{%
4888 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4889 \let\bbl@elt\relax
4890 \xdef\bbl@languages{%
4891 \bbl@languages\bbl@elt{#1}{#2}{}}}%
4892 \def\bbl@process@synonym#1{%
4893 \ifcase\count@

```

```

4894 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4895 \or
4896 \ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}}%
4897 \else
4898 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4899 \fi}
4900 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4901 \chardef\l@english\z@
4902 \chardef\l@USenglish\z@
4903 \chardef\bbl@last\z@
4904 \global\@namedef{bbl@hyphendata@0}{\hyphen.tex}}
4905 \gdef\bbl@languages{%
4906 \bbl@elt{english}{0}{\hyphen.tex}}}%
4907 \bbl@elt{USenglish}{0}{}}
4908 \else
4909 \global\let\bbl@languages@format\bbl@languages
4910 \def\bbl@elt#1#2#3#4{% Remove all except language 0
4911 \ifnum#2>\z@\else
4912 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4913 \fi}%
4914 \xdef\bbl@languages{\bbl@languages}%
4915 \fi
4916 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}} % Define flags
4917 \bbl@languages
4918 \openin\bbl@readstream=language.dat
4919 \ifeof\bbl@readstream
4920 \bbl@warning{I couldn't find language.dat. No additional\\%
4921 patterns loaded. Reported}%
4922 \else
4923 \loop
4924 \endlinechar\m@ne
4925 \read\bbl@readstream to \bbl@line
4926 \endlinechar\^^M
4927 \if T\ifeof\bbl@readstream F\fi T\relax
4928 \ifx\bbl@line\@empty\else
4929 \edef\bbl@line{\bbl@line\space\space\space}%
4930 \expandafter\bbl@process@line\bbl@line\relax
4931 \fi
4932 \repeat
4933 \fi
4934 \endgroup
4935 \bbl@trace{Macros for reading patterns files}
4936 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
4937 \ifx\babelcatcodetablenum\@undefined
4938 \ifx\newcatcodetable\@undefined
4939 \def\babelcatcodetablenum{5211}
4940 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4941 \else
4942 \newcatcodetable\babelcatcodetablenum
4943 \newcatcodetable\bbl@pattcodes
4944 \fi
4945 \else
4946 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4947 \fi
4948 \def\bbl@luapatterns#1#2{%
4949 \bbl@get@enc#1::\@@@
4950 \setbox\z@\hbox\bgroup
4951 \begingroup
4952 \savecatcodetable\babelcatcodetablenum\relax
4953 \initcatcodetable\bbl@pattcodes\relax
4954 \catcodetable\bbl@pattcodes\relax
4955 \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4956 \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~ =13

```

```

4957 \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4958 \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4959 \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4960 \catcode`\'=12 \catcode`\'=12 \catcode`\`=12
4961 \input #1\relax
4962 \catcodetable\babelcatcodetablenum\relax
4963 \endgroup
4964 \def\bbl@tempa{#2}%
4965 \ifx\bbl@tempa\empty\else
4966 \input #2\relax
4967 \fi
4968 \egroup}%
4969 \def\bbl@patterns@lua#1{%
4970 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4971 \csname l@#1\endcsname
4972 \edef\bbl@tempa{#1}%
4973 \else
4974 \csname l@#1:\f@encoding\endcsname
4975 \edef\bbl@tempa{#1:\f@encoding}%
4976 \fi\relax
4977 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4978 \@ifundefined{bbl@hyphendata@the\language}%
4979 {\def\bbl@elt##1###2###3###4{%
4980 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4981 \def\bbl@tempb{##3}%
4982 \ifx\bbl@tempb\empty\else % if not a synonymous
4983 \def\bbl@tempc{##3}{##4}}%
4984 \fi
4985 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4986 \fi}%
4987 \bbl@languages
4988 \@ifundefined{bbl@hyphendata@the\language}%
4989 {\bbl@info{No hyphenation patterns were set for\%
4990 language '\bbl@tempa'. Reported}}%
4991 {\expandafter\expandafter\expandafter\bbl@luapatterns
4992 \csname bbl@hyphendata@the\language\endcsname}}}%
4993 \endinput\fi
4994 % Here ends \ifx\AddBabelHook\undefined
4995 % A few lines are only read by hyphen.cfg
4996 \ifx\DisableBabelHook\undefined
4997 \AddBabelHook{luatex}{everylanguage}{%
4998 \def\process@language##1##2##3{%
4999 \def\process@line####1####2 ####3 ####4 {}}}
5000 \AddBabelHook{luatex}{loadpatterns}{%
5001 \input #1\relax
5002 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5003 {{#1}}}%
5004 \AddBabelHook{luatex}{loadexceptions}{%
5005 \input #1\relax
5006 \def\bbl@tempb##1##2{{##1}{#1}}%
5007 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5008 {\expandafter\expandafter\expandafter\bbl@tempb
5009 \csname bbl@hyphendata@the\language\endcsname}}
5010 \endinput\fi
5011 % Here stops reading code for hyphen.cfg
5012 % The following is read the 2nd time it's loaded
5013 \begingroup % TODO - to a lua file
5014 \catcode`\%=12
5015 \catcode`\'=12
5016 \catcode`\`=12
5017 \catcode`\:=12
5018 \directlua{
5019 Babel = Babel or {}

```

```

5020 function Babel.bytes(line)
5021   return line:gsub("(.)",
5022     function (chr) return unicode.utf8.char(string.byte(chr)) end)
5023 end
5024 function Babel.begin_process_input()
5025   if luatexbase and luatexbase.add_to_callback then
5026     luatexbase.add_to_callback('process_input_buffer',
5027       Babel.bytes, 'Babel.bytes')
5028   else
5029     Babel.callback = callback.find('process_input_buffer')
5030     callback.register('process_input_buffer', Babel.bytes)
5031   end
5032 end
5033 function Babel.end_process_input ()
5034   if luatexbase and luatexbase.remove_from_callback then
5035     luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5036   else
5037     callback.register('process_input_buffer', Babel.callback)
5038   end
5039 end
5040 function Babel.addpatterns(pp, lg)
5041   local lg = lang.new(lg)
5042   local pats = lang.patterns(lg) or ''
5043   lang.clear_patterns(lg)
5044   for p in pp:gmatch('[^%s]+') do
5045     ss = ''
5046     for i in string.utfcharacters(p:gsub('%d', '')) do
5047       ss = ss .. '%d?' .. i
5048     end
5049     ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
5050     ss = ss:gsub('%.%%d%?$', '%%.')
5051     pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5052     if n == 0 then
5053       tex.sprint(
5054         [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5055         .. p .. [[{}]])
5056       pats = pats .. ' ' .. p
5057     else
5058       tex.sprint(
5059         [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5060         .. p .. [[{}]])
5061     end
5062   end
5063   lang.patterns(lg, pats)
5064 end
5065 Babel.characters = Babel.characters or {}
5066 Babel.ranges = Babel.ranges or {}
5067 function Babel.hlist_has_bidi(head)
5068   local has_bidi = false
5069   local ranges = Babel.ranges
5070   for item in node.traverse(head) do
5071     if item.id == node.id'glyph' then
5072       local itemchar = item.char
5073       local chardata = Babel.characters[itemchar]
5074       local dir = chardata and chardata.d or nil
5075       if not dir then
5076         for nn, et in ipairs(ranges) do
5077           if itemchar < et[1] then
5078             break
5079           elseif itemchar <= et[2] then
5080             dir = et[3]
5081             break
5082           end
5083         end
5084       end
5085     end
5086   end
5087 end

```

```

5083         end
5084     end
5085     if dir and (dir == 'al' or dir == 'r') then
5086         has_bidi = true
5087     end
5088 end
5089 end
5090 return has_bidi
5091 end
5092 function Babel.set_chrnges_b (script, chrng)
5093     if chrng == '' then return end
5094     texio.write('Replacing ' .. script .. ' script ranges')
5095     Babel.script_blocks[script] = {}
5096     for s, e in string.gmatch(chrng..' ', '(-)%.%(-)%s') do
5097         table.insert(
5098             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5099     end
5100 end
5101 }
5102 \endgroup
5103 \ifx\newattribute\@undefined\else
5104     \newattribute\bbl@attr@locale
5105     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5106     \AddBabelHook{luatex}{beforeextras}{%
5107         \setattribute\bbl@attr@locale\localeid}
5108 \fi
5109 \def\BabelStringsDefault{unicode}
5110 \let\luabbl@stop\relax
5111 \AddBabelHook{luatex}{encodedcommands}{%
5112     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5113     \ifx\bbl@tempa\bbl@tempb\else
5114         \directlua{Babel.begin_process_input()}%
5115         \def\luabbl@stop{%
5116             \directlua{Babel.end_process_input()}}%
5117     \fi}%
5118 \AddBabelHook{luatex}{stopcommands}{%
5119     \luabbl@stop
5120     \let\luabbl@stop\relax}
5121 \AddBabelHook{luatex}{patterns}{%
5122     \@ifundefined{bbl@hyphendata@the\language}%
5123     {\def\bbl@elt##1##2##3##4{%
5124         \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5125         \def\bbl@tempb{##3}%
5126         \ifx\bbl@tempb\@empty\else % if not a synonymous
5127             \def\bbl@tempc{##3}##4}%
5128         \fi
5129         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5130     \fi}%
5131     \bbl@languages
5132     \@ifundefined{bbl@hyphendata@the\language}%
5133     {\bbl@info{No hyphenation patterns were set for\%
5134         language '#2'. Reported}}%
5135     {\expandafter\expandafter\expandafter\bbl@luapatterns
5136         \csname bbl@hyphendata@the\language\endcsname}}}%
5137 \@ifundefined{bbl@patterns@}{}%
5138 \begingroup
5139     \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5140     \ifin@\else
5141         \ifx\bbl@patterns@\@empty\else
5142             \directlua{ Babel.addpatterns(
5143                 [[\bbl@patterns@]], \number\language) }%
5144         \fi
5145     \@ifundefined{bbl@patterns@#1}%

```

```

5146      \@empty
5147      {\directlua{ Babel.addpatterns(
5148          [[\space\csname bbl@patterns@#1\endcsname]],
5149          \number\language) }}%
5150      \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5151      \fi
5152      \endgroup}%
5153      \bbl@exp{%
5154      \bbl@ifunset{\bbl@prehc@\languagename}{}%
5155      {\bbl@ifblank{\bbl@cs{\prehc@\languagename}}{}}%
5156      {\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.

```

5157 \@onlypreamble\babelpatterns
5158 \AtEndOfPackage{%
5159   \newcommand\babelpatterns[2][\@empty]{%
5160     \ifx\bbl@patterns@\relax
5161       \let\bbl@patterns@\@empty
5162     \fi
5163     \ifx\bbl@pttnlist@\empty\else
5164       \bbl@warning{%
5165         You must not intermingle \string\selectlanguage\space and\\%
5166         \string\babelpatterns\space or some patterns will not\\%
5167         be taken into account. Reported}%
5168     \fi
5169     \ifx\@empty#1%
5170       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5171     \else
5172       \edef\bbl@tempb{\zap@space#1 \@empty}%
5173       \bbl@for\bbl@tempa\bbl@tempb{%
5174         \bbl@fixname\bbl@tempa
5175         \bbl@iflanguage\bbl@tempa{%
5176           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5177             \@ifundefined{\bbl@patterns@\bbl@tempa}%
5178             \@empty
5179             {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5180             #2}}}%
5181     \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.

```

5182 % TODO - to a lua file
5183 \directlua{
5184   Babel = Babel or {}
5185   Babel.linebreaking = Babel.linebreaking or {}
5186   Babel.linebreaking.before = {}
5187   Babel.linebreaking.after = {}
5188   Babel.locale = {} % Free to use, indexed by \localeid
5189   function Babel.linebreaking.add_before(func)
5190     tex.print([[ \noexpand\csname bbl@luahyphenate\endcsname ]])
5191     table.insert(Babel.linebreaking.before, func)
5192   end
5193   function Babel.linebreaking.add_after(func)
5194     tex.print([[ \noexpand\csname bbl@luahyphenate\endcsname ]])
5195     table.insert(Babel.linebreaking.after, func)
5196   end

```



```

5197 }
5198 \def\bbl@intraspace#1 #2 #3\@{%
5199   \directlua{
5200     Babel = Babel or {}
5201     Babel.intraspaces = Babel.intraspaces or {}
5202     Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5203       {b = #1, p = #2, m = #3}
5204     Babel.locale_props[\the\localeid].intraspace = %
5205       {b = #1, p = #2, m = #3}
5206   }}
5207 \def\bbl@intrapenalty#1\@{%
5208   \directlua{
5209     Babel = Babel or {}
5210     Babel.intrapenalties = Babel.intrapenalties or {}
5211     Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5212     Babel.locale_props[\the\localeid].intrapenalty = #1
5213   }}
5214 \begingroup
5215 \catcode`\%=12
5216 \catcode`\^=14
5217 \catcode`\'=12
5218 \catcode`\~=12
5219 \gdef\bbl@seaintraspace{^
5220   \let\bbl@seaintraspace\relax
5221   \directlua{
5222     Babel = Babel or {}
5223     Babel.sea_enabled = true
5224     Babel.sea_ranges = Babel.sea_ranges or {}
5225     function Babel.set_chrngs (script, chrng)
5226       local c = 0
5227       for s, e in string.gmatch(chrng..' ', '(.-%.-%s') do
5228         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5229         c = c + 1
5230       end
5231     end
5232     function Babel.sea_disc_to_space (head)
5233       local sea_ranges = Babel.sea_ranges
5234       local last_char = nil
5235       local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5236       for item in node.traverse(head) do
5237         local i = item.id
5238         if i == node.id'glyph' then
5239           last_char = item
5240         elseif i == 7 and item.subtype == 3 and last_char
5241           and last_char.char > 0x0C99 then
5242           quad = font.getfont(last_char.font).size
5243           for lg, rg in pairs(sea_ranges) do
5244             if last_char.char > rg[1] and last_char.char < rg[2] then
5245               lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyril1
5246               local intraspace = Babel.intraspaces[lg]
5247               local intrapenalty = Babel.intrapenalties[lg]
5248               local n
5249               if intrapenalty ~= 0 then
5250                 n = node.new(14, 0)      ^% penalty
5251                 n.penalty = intrapenalty
5252                 node.insert_before(head, item, n)
5253               end
5254               n = node.new(12, 13)      ^% (glue, spaceskip)
5255               node.setglue(n, intraspace.b * quad,
5256                 intraspace.p * quad,
5257                 intraspace.m * quad)
5258               node.insert_before(head, item, n)
5259               node.remove(head, item)

```

```

5260         end
5261     end
5262 end
5263 end
5264 end
5265 }^^
5266 \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.

```

5267 \catcode`\%=14
5268 \gdef\bbl@cjkintraspacespace{%
5269   \let\bbl@cjkintraspacespace\relax
5270   \directlua{
5271     Babel = Babel or {}
5272     require('babel-data-cjk.lua')
5273     Babel.cjk_enabled = true
5274     function Babel.cjk_linebreak(head)
5275       local GLYPH = node.id'glyph'
5276       local last_char = nil
5277       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5278       local last_class = nil
5279       local last_lang = nil
5280
5281       for item in node.traverse(head) do
5282         if item.id == GLYPH then
5283
5284           local lang = item.lang
5285
5286           local LOCALE = node.get_attribute(item,
5287             Babel.attr_locale)
5288           local props = Babel.locale_props[LOCALE]
5289
5290           local class = Babel.cjk_class[item.char].c
5291
5292           if props.cjk_quotes and props.cjk_quotes[item.char] then
5293             class = props.cjk_quotes[item.char]
5294           end
5295
5296           if class == 'cp' then class = 'cl' end % )] as CL
5297           if class == 'id' then class = 'I' end
5298
5299           local br = 0
5300           if class and last_class and Babel.cjk_breaks[last_class][class] then
5301             br = Babel.cjk_breaks[last_class][class]
5302           end
5303
5304           if br == 1 and props.linebreak == 'c' and
5305             lang ~= \the\l@nohyphenation\space and
5306             last_lang ~= \the\l@nohyphenation then
5307             local intrapenalty = props.intrapenalty
5308             if intrapenalty ~= 0 then
5309               local n = node.new(14, 0)      % penalty
5310               n.penalty = intrapenalty
5311               node.insert_before(head, item, n)
5312             end
5313             local intraspacespace = props.intraspacespace

```

```

5314         local n = node.new(12, 13)      % (glue, spaceskip)
5315         node.setglue(n, intraspace.b * quad,
5316                     intraspace.p * quad,
5317                     intraspace.m * quad)
5318         node.insert_before(head, item, n)
5319     end
5320
5321     if font.getfont(item.font) then
5322         quad = font.getfont(item.font).size
5323     end
5324     last_class = class
5325     last_lang = lang
5326     else % if penalty, glue or anything else
5327         last_class = nil
5328     end
5329 end
5330 lang.hyphenate(head)
5331 end
5332 }%
5333 \bbl@luahyphenate}
5334 \gdef\bbl@luahyphenate{%
5335 \let\bbl@luahyphenate\relax
5336 \directlua{
5337     luatexbase.add_to_callback('hyphenate',
5338     function (head, tail)
5339         if Babel.linebreaking.before then
5340             for k, func in ipairs(Babel.linebreaking.before) do
5341                 func(head)
5342             end
5343         end
5344         if Babel.cjk_enabled then
5345             Babel.cjk_linebreak(head)
5346         end
5347         lang.hyphenate(head)
5348         if Babel.linebreaking.after then
5349             for k, func in ipairs(Babel.linebreaking.after) do
5350                 func(head)
5351             end
5352         end
5353         if Babel.sea_enabled then
5354             Babel.sea_disc_to_space(head)
5355         end
5356     end,
5357     'Babel.hyphenate')
5358 }
5359 }
5360 \endgroup
5361 \def\bbl@provide@intraspace{%
5362 \bbl@ifunset\bbl@intsp@\languagename}{}%
5363 {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
5364 \bbl@xin@{/c}{/\bbl@c1{lnbrk}}}%
5365 \ifin@ % cjk
5366 \bbl@cjk intraspace
5367 \directlua{
5368     Babel = Babel or {}
5369     Babel.locale_props = Babel.locale_props or {}
5370     Babel.locale_props[\the\localeid].linebreak = 'c'
5371 }%
5372 \bbl@exp{\bbl@intraspace\bbl@c1{intsp}\bbl@intsp}%
5373 \ifx\bbl@KVP@intrapenalty\@nnil
5374 \bbl@intrapenalty0\@nnil
5375 \fi
5376 \else % sea

```

```

5377      \bbl@seaintraspace
5378      \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\bbl@@}%
5379      \directlua{
5380          Babel = Babel or {}
5381          Babel.sea_ranges = Babel.sea_ranges or {}
5382          Babel.set_chranges('\bbl@cl{sbcpr}',
5383                          '\bbl@cl{chrng}')
5384      }%
5385      \ifx\bbl@KVP@intrapenalty\@nnil
5386          \bbl@intrapenalty0@@
5387      \fi
5388      \fi
5389      \fi
5390      \ifx\bbl@KVP@intrapenalty\@nnil\else
5391          \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty@@
5392      \fi}}

```

## 12.6 Arabic justification

```

5393 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5394 \def\bblar@chars{%
5395     0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5396     0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5397     0640,0641,0642,0643,0644,0645,0646,0647,0649}
5398 \def\bblar@elongated{%
5399     0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5400     063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5401     0649,064A}
5402 \begingroup
5403   \catcode`\_ =11 \catcode`\:=11
5404   \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5405 \endgroup
5406 \gdef\bbl@arabicjust{%
5407   \let\bbl@arabicjust\relax
5408   \newattribute\bblar@kashida
5409   \directlua{ Babel.attr_kashida = luatexbase.registernumber'bbblar@kashida' }%
5410   \bblar@kashida=\z@
5411   \bbl@patchfont{\bbl@parsejalt}}%
5412   \directlua{
5413       Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5414       Babel.arabic.elong_map[\the\localeid] = {}
5415       luatexbase.add_to_callback('post_linebreak_filter',
5416           Babel.arabic.justify, 'Babel.arabic.justify')
5417       luatexbase.add_to_callback('hpack_filter',
5418           Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5419   }%
5420 % Save both node lists to make replacement. TODO. Save also widths to
5421 % make computations
5422 \def\bblar@fetchjalt#1#2#3#4{%
5423   \bbl@exp{\bbl@foreach{#1}}{%
5424     \bbl@ifunset{bbblar@JE##1}%
5425     {\setbox\z@\hbox{^^^200d\char"##1#2}}%
5426     {\setbox\z@\hbox{^^^200d\char"@nameuse{bbblar@JE##1}#2}}%
5427   \directlua{%
5428     local last = nil
5429     for item in node.traverse(tex.box[0].head) do
5430       if item.id == node.id'glyph' and item.char > 0x600 and
5431         not (item.char == 0x200D) then
5432         last = item
5433       end
5434     end
5435     Babel.arabic.#3['##1#4'] = last.char
5436   }}

```

```

5437 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5438 % perhaps other tables (falt?, csw?). What about kaf? And diacritic
5439 % positioning?
5440 \gdef\bbl@parsejalt{%
5441   \ifx\addfontfeature\undefined\else
5442     \bbl@xin@{/e}{/\bbl@c1{lnbrk}}%
5443     \ifin@
5444       \directlua{%
5445         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5446           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5447           tex.print([[string\curname\space bbl@parsejalti\endcurname]])
5448         end
5449       }%
5450     \fi
5451   \fi}
5452 \gdef\bbl@parsejalti{%
5453   \begingroup
5454     \let\bbl@parsejalt\relax % To avoid infinite loop
5455     \edef\bbl@tempb{\fontid\font}%
5456     \bblar@nofswarn
5457     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5458     \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5459     \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5460     \addfontfeature{RawFeature+=jalt}%
5461     % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5462     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5463     \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5464     \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5465     \directlua{%
5466       for k, v in pairs(Babel.arabic.from) do
5467         if Babel.arabic.dest[k] and
5468           not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5469           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5470             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5471         end
5472       end
5473     }%
5474   \endgroup}
5475 %
5476 \begingroup
5477 \catcode`#=11
5478 \catcode`~ =11
5479 \directlua{
5480
5481 Babel.arabic = Babel.arabic or {}
5482 Babel.arabic.from = {}
5483 Babel.arabic.dest = {}
5484 Babel.arabic.justify_factor = 0.95
5485 Babel.arabic.justify_enabled = true
5486
5487 function Babel.arabic.justify(head)
5488   if not Babel.arabic.justify_enabled then return head end
5489   for line in node.traverse_id(node.id'hlist', head) do
5490     Babel.arabic.justify_hlist(head, line)
5491   end
5492   return head
5493 end
5494
5495 function Babel.arabic.justify_hbox(head, gc, size, pack)
5496   local has_inf = false
5497   if Babel.arabic.justify_enabled and pack == 'exactly' then
5498     for n in node.traverse_id(12, head) do
5499       if n.stretch_order > 0 then has_inf = true end

```

```

5500     end
5501     if not has_inf then
5502         Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5503     end
5504 end
5505 return head
5506 end
5507
5508 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5509     local d, new
5510     local k_list, k_item, pos_inline
5511     local width, width_new, full, k_curr, wt_pos, goal, shift
5512     local subst_done = false
5513     local elong_map = Babel.arabic.elong_map
5514     local last_line
5515     local GLYPH = node.id'glyph'
5516     local KASHIDA = Babel.attr_kashida
5517     local LOCALE = Babel.attr_locale
5518
5519     if line == nil then
5520         line = {}
5521         line.glue_sign = 1
5522         line.glue_order = 0
5523         line.head = head
5524         line.shift = 0
5525         line.width = size
5526     end
5527
5528     % Exclude last line. todo. But-- it discards one-word lines, too!
5529     % ? Look for glue = 12:15
5530     if (line.glue_sign == 1 and line.glue_order == 0) then
5531         elongs = {}      % Stores elongated candidates of each line
5532         k_list = {}      % And all letters with kashida
5533         pos_inline = 0   % Not yet used
5534
5535         for n in node.traverse_id(GLYPH, line.head) do
5536             pos_inline = pos_inline + 1 % To find where it is. Not used.
5537
5538             % Elongated glyphs
5539             if elong_map then
5540                 local locale = node.get_attribute(n, LOCALE)
5541                 if elong_map[locale] and elong_map[locale][n.font] and
5542                     elong_map[locale][n.font][n.char] then
5543                     table.insert(elongs, {node = n, locale = locale} )
5544                     node.set_attribute(n.prev, KASHIDA, 0)
5545                 end
5546             end
5547
5548             % Tatwil
5549             if Babel.kashida_wts then
5550                 local k_wt = node.get_attribute(n, KASHIDA)
5551                 if k_wt > 0 then % todo. parameter for multi inserts
5552                     table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5553                 end
5554             end
5555
5556         end % of node.traverse_id
5557
5558         if #elongs == 0 and #k_list == 0 then goto next_line end
5559         full = line.width
5560         shift = line.shift
5561         goal = full * Babel.arabic.justify_factor % A bit crude
5562         width = node.dimensions(line.head)      % The 'natural' width

```

```

5563
5564 % == Elongated ==
5565 % Original idea taken from 'chickenize'
5566 while (#elongs > 0 and width < goal) do
5567     subst_done = true
5568     local x = #elongs
5569     local curr = elongs[x].node
5570     local oldchar = curr.char
5571     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5572     width = node.dimensions(line.head) % Check if the line is too wide
5573     % Substitute back if the line would be too wide and break:
5574     if width > goal then
5575         curr.char = oldchar
5576         break
5577     end
5578     % If continue, pop the just substituted node from the list:
5579     table.remove(elongs, x)
5580 end
5581
5582 % == Tatwil ==
5583 if #k_list == 0 then goto next_line end
5584
5585 width = node.dimensions(line.head) % The 'natural' width
5586 k_curr = #k_list
5587 wt_pos = 1
5588
5589 while width < goal do
5590     subst_done = true
5591     k_item = k_list[k_curr].node
5592     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5593         d = node.copy(k_item)
5594         d.char = 0x0640
5595         line.head, new = node.insert_after(line.head, k_item, d)
5596         width_new = node.dimensions(line.head)
5597         if width > goal or width == width_new then
5598             node.remove(line.head, new) % Better compute before
5599             break
5600         end
5601         width = width_new
5602     end
5603     if k_curr == 1 then
5604         k_curr = #k_list
5605         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5606     else
5607         k_curr = k_curr - 1
5608     end
5609 end
5610
5611 ::next_line::
5612
5613 % Must take into account marks and ins, see luatex manual.
5614 % Have to be executed only if there are changes. Investigate
5615 % what's going on exactly.
5616 if subst_done and not gc then
5617     d = node.hpack(line.head, full, 'exactly')
5618     d.shift = shift
5619     node.insert_before(head, line, d)
5620     node.remove(head, line)
5621 end
5622 end % if process line
5623 end
5624 }
5625 \endgroup

```

```
5626 \fi\fi % Arabic just block
```

## 12.7 Common stuff

```
5627 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5628 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ccheckstdfonts}
5629 \DisableBabelHook{babel-fontspec}
5630 <<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.

```
5631 % TODO - to a lua file
5632 \directlua{
5633 Babel.script_blocks = {
5634   ['dflt'] = {},
5635   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5636               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5637   ['Armn'] = {{0x0530, 0x058F}},
5638   ['Beng'] = {{0x0980, 0x09FF}},
5639   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5640   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5641   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5642               {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5643   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5644   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5645               {0xAB00, 0xAB2F}},
5646   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5647   % Don't follow strictly Unicode, which places some Coptic letters in
5648   % the 'Greek and Coptic' block
5649   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5650   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5651               {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5652               {0xF900, 0FAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5653               {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5654               {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5655               {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5656   ['Hebr'] = {{0x0590, 0x05FF}},
5657   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5658               {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5659   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5660   ['Knda'] = {{0x0C80, 0x0CFF}},
5661   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5662               {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5663               {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5664   ['Lao'] = {{0x0E80, 0x0EFF}},
5665   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5666               {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5667               {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5668   ['Mahj'] = {{0x11150, 0x1117F}},
5669   ['Mlym'] = {{0x0D00, 0x0D7F}},
5670   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5671   ['Orya'] = {{0x0B00, 0x0B7F}},
5672   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5673   ['Syr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5674   ['Taml'] = {{0x0B80, 0x0BFF}},
5675   ['Telu'] = {{0x0C00, 0x0C7F}},
5676   ['Tfng'] = {{0x2D30, 0x2D7F}},
```



```

5677 ['Thai'] = {{0x0E00, 0x0E7F}},
5678 ['Tibt'] = {{0x0F00, 0x0FFF}},
5679 ['Vaii'] = {{0xA500, 0xA63F}},
5680 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5681 }
5682
5683 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5684 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5685 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5686
5687 function Babel.locale_map(head)
5688   if not Babel.locale_mapped then return head end
5689
5690   local LOCALE = Babel.attr_locale
5691   local GLYPH = node.id('glyph')
5692   local inmath = false
5693   local toloc_save
5694   for item in node.traverse(head) do
5695     local toloc
5696     if not inmath and item.id == GLYPH then
5697       % Optimization: build a table with the chars found
5698       if Babel.chr_to_loc[item.char] then
5699         toloc = Babel.chr_to_loc[item.char]
5700       else
5701         for lc, maps in pairs(Babel.loc_to_scr) do
5702           for _, rg in pairs(maps) do
5703             if item.char >= rg[1] and item.char <= rg[2] then
5704               Babel.chr_to_loc[item.char] = lc
5705               toloc = lc
5706               break
5707             end
5708           end
5709         end
5710       end
5711       % Now, take action, but treat composite chars in a different
5712       % fashion, because they 'inherit' the previous locale. Not yet
5713       % optimized.
5714       if not toloc and
5715         (item.char >= 0x0300 and item.char <= 0x036F) or
5716         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5717         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5718         toloc = toloc_save
5719       end
5720       if toloc and Babel.locale_props[toloc] and
5721         Babel.locale_props[toloc].letters and
5722         tex.getcatcode(item.char) \string~= 11 then
5723         toloc = nil
5724       end
5725       if toloc and toloc > -1 then
5726         if Babel.locale_props[toloc].lg then
5727           item.lang = Babel.locale_props[toloc].lg
5728           node.set_attribute(item, LOCALE, toloc)
5729         end
5730         if Babel.locale_props[toloc]['/'..item.font] then
5731           item.font = Babel.locale_props[toloc]['/'..item.font]
5732         end
5733         toloc_save = toloc
5734       end
5735     elseif not inmath and item.id == 7 then % Apply recursively
5736       item.replace = item.replace and Babel.locale_map(item.replace)
5737       item.pre = item.pre and Babel.locale_map(item.pre)
5738       item.post = item.post and Babel.locale_map(item.post)
5739     elseif item.id == node.id'math' then

```

```

5740     inmath = (item.subtype == 0)
5741   end
5742 end
5743 return head
5744 end
5745 }

```

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

```

5746 \newcommand\babelcharproperty[1]{%
5747   \count@=#1\relax
5748   \ifvmode
5749     \expandafter\bbl@chprop
5750   \else
5751     \bbl@error{\string\babelcharproperty\space can be used only in\%
5752       vertical mode (preamble or between paragraphs)}%
5753     {See the manual for futher info}%
5754   \fi}
5755 \newcommand\bbl@chprop[3][\the\count@]{%
5756   \@tempcnta=#1\relax
5757   \bbl@ifunset{\bbl@chprop@#2}%
5758   {\bbl@error{No property named '#2'. Allowed values are\%
5759     direction (bc), mirror (bmg), and linebreak (lb)}%
5760     {See the manual for futher info}}%
5761   {}%
5762   \loop
5763     \bbl@cs{chprop@#2}{#3}%
5764     \ifnum\count@<\@tempcnta
5765       \advance\count@\@ne
5766     \repeat}
5767 \def\bbl@chprop@direction#1{%
5768   \directlua{
5769     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5770     Babel.characters[\the\count@]['d'] = '#1'
5771   }}
5772 \let\bbl@chprop@bc\bbl@chprop@direction
5773 \def\bbl@chprop@mirror#1{%
5774   \directlua{
5775     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5776     Babel.characters[\the\count@]['m'] = '\number#1'
5777   }}
5778 \let\bbl@chprop@bmg\bbl@chprop@mirror
5779 \def\bbl@chprop@linebreak#1{%
5780   \directlua{
5781     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5782     Babel.cjk_characters[\the\count@]['c'] = '#1'
5783   }}
5784 \let\bbl@chprop@lb\bbl@chprop@linebreak
5785 \def\bbl@chprop@locale#1{%
5786   \directlua{
5787     Babel.chr_to_loc = Babel.chr_to_loc or {}
5788     Babel.chr_to_loc[\the\count@] =
5789       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
5790   }}

```

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.

```

5791 \directlua{
5792   Babel.nohyphenation = \the\l@nohyphenation
5793 }

```

Now the  $\TeX$  high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the `{n}` syntax. For example, `pre={1}{1}-` becomes `function(m) return m[1]..m[1]..'-' end`, where `m` are the matches returned after

applying the pattern. With a mapped capture the functions are similar to  
function(m) return Babel.capt\_map(m[1],1) end, where the last argument identifies the  
mapping to be applied to m[1]. The way it is carried out is somewhat tricky, but the effect 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).

```

5794 \begingroup
5795 \catcode`\~ = 12
5796 \catcode`\% = 12
5797 \catcode`\& = 14
5798 \catcode`\| = 12
5799 \gdef\babelprehyphenation{&
5800 \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}}
5801 \gdef\babelposthyphenation{&
5802 \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}}
5803 \gdef\bbl@postlinebreak{\bbl@settransform{2}[]} & WIP
5804 \gdef\bbl@settransform#1[#2]#3#4#5{&
5805 \ifcase#1
5806 \bbl@activateprehyphen
5807 \or
5808 \bbl@activateposthyphen
5809 \fi
5810 \begingroup
5811 \def\babeltempa{\bbl@add@list\babeltempb}&
5812 \let\babeltempb\empty
5813 \def\bbl@tempa{#5}&
5814 \bbl@replace\bbl@tempa{,}{,}& TODO. Ugly trick to preserve {}
5815 \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&
5816 \bbl@ifsamestring{##1}{remove}&
5817 {\bbl@add@list\babeltempb{nil}}&
5818 {\directlua{
5819 local rep = [=##1]=]
5820 rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5821 rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5822 rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5823 if #1 == 0 or #1 == 2 then
5824 rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5825 'space = {' .. '%2, %3, %4' .. '}')
5826 rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5827 'spacefactor = {' .. '%2, %3, %4' .. '}')
5828 rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
5829 else
5830 rep = rep:gsub('(no)%s*=%s*([^\s,]*)', Babel.capture_func)
5831 rep = rep:gsub('(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5832 rep = rep:gsub('(post)%s*=%s*([^\s,]*)', Babel.capture_func)
5833 end
5834 tex.print([[\\string\babeltempa{}} .. rep .. [{}]])
5835 }}&
5836 \bbl@foreach\babeltempb{&
5837 \bbl@forkv{##1}{&
5838 \in@{,###1,}{,nil,step,data,remove,insert,string,no,pre,&
5839 no,post,penalty,kashida,space,spacefactor,}&
5840 \ifin\else
5841 \bbl@error
5842 {Bad option '###1' in a transform.\\&
5843 I'll ignore it but expect more errors}&
5844 {See the manual for further info.}&
5845 \fi}}&
5846 \let\bbl@kv@attribute\relax
5847 \let\bbl@kv@label\relax
5848 \bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}&
5849 \ifx\bbl@kv@attribute\relax\else
5850 \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&

```

```

5851 \fi
5852 \directlua{
5853     local lbkr = Babel.linebreaking.replacements[#1]
5854     local u = unicode.utf8
5855     local id, attr, label
5856     if #1 == 0 or #1 == 2 then
5857         id = \the\csname bbl@id@@#3\endcsname\space
5858     else
5859         id = \the\csname l@#3\endcsname\space
5860     end
5861     \ifx\bbl@kv@attribute\relax
5862         attr = -1
5863     \else
5864         attr = luatexbase.registernumber'\bbl@kv@attribute'
5865     \fi
5866     \ifx\bbl@kv@label\relax\else %% Same refs:
5867         label = [==[\bbl@kv@label]==]
5868     \fi
5869     %% Convert pattern:
5870     local patt = string.gsub([==[#4]==], '%s', '')
5871     if #1 == 0 or #1 == 2 then
5872         patt = string.gsub(patt, '|', ' ')
5873     end
5874     if not u.find(patt, '()', nil, true) then
5875         patt = '()' .. patt .. '()'
5876     end
5877     if #1 == 1 then
5878         patt = string.gsub(patt, '%(%)%^', '^()')
5879         patt = string.gsub(patt, '%$$(%)', '()$')
5880     end
5881     patt = u.gsub(patt, '{(.)}',
5882         function (n)
5883             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5884         end)
5885     patt = u.gsub(patt, '{(%x%x%x%x+)}',
5886         function (n)
5887             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
5888         end)
5889     lbkr[id] = lbkr[id] or {}
5890     table.insert(lbkr[id],
5891         { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
5892 }%%
5893 \endgroup}
5894 \endgroup
5895 \def\bbl@activateposthyphen{%
5896     \let\bbl@activateposthyphen\relax
5897     \directlua{
5898         require('babel-transforms.lua')
5899         Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5900     }}
5901 \def\bbl@activateprehyphen{%
5902     \let\bbl@activateprehyphen\relax
5903     \directlua{
5904         require('babel-transforms.lua')
5905         Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5906     }}

```

## 12.9 Bidi

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

```

5907 \def\bbl@activate@preotf{%

```

```

5908 \let\bbl@activate@preotf\relax % only once
5909 \directlua{
5910   Babel = Babel or {}
5911   %
5912   function Babel.pre_otfload_v(head)
5913     if Babel.numbers and Babel.digits_mapped then
5914       head = Babel.numbers(head)
5915     end
5916     if Babel.bidi_enabled then
5917       head = Babel.bidi(head, false, dir)
5918     end
5919     return head
5920   end
5921   %
5922   function Babel.pre_otfload_h(head, gc, sz, pt, dir)
5923     if Babel.numbers and Babel.digits_mapped then
5924       head = Babel.numbers(head)
5925     end
5926     if Babel.bidi_enabled then
5927       head = Babel.bidi(head, false, dir)
5928     end
5929     return head
5930   end
5931   %
5932   luatexbase.add_to_callback('pre_linebreak_filter',
5933     Babel.pre_otfload_v,
5934     'Babel.pre_otfload_v',
5935     luatexbase.priority_in_callback('pre_linebreak_filter',
5936       'luaotfload.node_processor') or nil)
5937   %
5938   luatexbase.add_to_callback('hpack_filter',
5939     Babel.pre_otfload_h,
5940     'Babel.pre_otfload_h',
5941     luatexbase.priority_in_callback('hpack_filter',
5942       'luaotfload.node_processor') or nil)
5943 }

```

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=`.

```

5944 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5945   \let\bbl@beforeforeign\leavevmode
5946   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5947   \RequirePackage{luatexbase}
5948   \bbl@activate@preotf
5949   \directlua{
5950     require('babel-data-bidi.lua')
5951     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
5952       require('babel-bidi-basic.lua')
5953     \or
5954       require('babel-bidi-basic-r.lua')
5955     \fi}
5956   % TODO - to locale_props, not as separate attribute
5957   \newattribute\bbl@attr@dir
5958   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
5959   % TODO. I don't like it, hackish:
5960   \bbl@exp{\output{\bodydir\pagedir\the\output}}
5961   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5962 \fi\fi
5963 \chardef\bbl@thetextdir\z@
5964 \chardef\bbl@thepardir\z@
5965 \def\bbl@getluadir#1{%
5966   \directlua{

```

```

5967   if tex.#1dir == 'TLT' then
5968       tex.sprint('0')
5969   elseif tex.#1dir == 'TRT' then
5970       tex.sprint('1')
5971   end}}
5972 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
5973   \ifcase#3\relax
5974   \ifcase\bbl@getluadir{#1}\relax\else
5975       #2 TLT\relax
5976   \fi
5977   \else
5978   \ifcase\bbl@getluadir{#1}\relax
5979       #2 TRT\relax
5980   \fi
5981   \fi}
5982 \def\bbl@thedir{0}
5983 \def\bbl@textdir#1{%
5984   \bbl@setluadir{text}\textdir{#1}%
5985   \chardef\bbl@thetextdir#1\relax
5986   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*3+#1}%
5987   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
5988 \def\bbl@pardir#1{%
5989   \bbl@setluadir{par}\pardir{#1}%
5990   \chardef\bbl@thepardir#1\relax}
5991 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
5992 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
5993 \def\bbl@dirparastext{\pardir\the\textdir\relax}%   %%%
5994 %
5995 \ifnum\bbl@bidimode>\z@
5996   \def\bbl@insidemath{0}%
5997   \def\bbl@everymath{\def\bbl@insidemath{1}}
5998   \def\bbl@everydisplay{\def\bbl@insidemath{2}}
5999   \frozen@everymath\expandafter{%
6000     \expandafter\bbl@everymath\the\frozen@everymath}
6001   \frozen@everydisplay\expandafter{%
6002     \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6003   \AtBeginDocument{
6004     \directlua{
6005       function Babel.math_box_dir(head)
6006         if not (token.get_macro('bbl@insidemath') == '0') then
6007           if Babel.hlist_has_bidi(head) then
6008             local d = node.new(node.id'dir')
6009             d.dir = '+TRT'
6010             node.insert_before(head, node.has_glyph(head), d)
6011             for item in node.traverse(head) do
6012               node.set_attribute(item,
6013                 Babel.attr_dir, token.get_macro('bbl@thedir'))
6014             end
6015           end
6016         end
6017         return head
6018       end
6019       luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6020         "Babel.math_box_dir", 0)
6021     } }%
6022 \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, `dcolum` still fails.

```

6023 \bbl@trace{Redefinitions for bidi layout}
6024 %
6025 \langle *More package options\rangle \equiv
6026 \chardef\bbl@eqnpos\z@
6027 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6028 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6029 \rangle /More package options\rangle
6030 %
6031 \def\BabelNoAMSMath{\let\bbl@noamsmath\relax}
6032 \ifnum\bbl@bidimode>\z@
6033   \ifx\matheqdirmode\undefined\else
6034     \matheqdirmode\@ne
6035     \fi
6036     \let\bbl@eqnodir\relax
6037     \def\bbl@eqdel{()}
6038     \def\bbl@eqnum{%
6039       {\normalfont\normalcolor
6040         \expandafter\@firstoftwo\bbl@eqdel
6041         \theequation
6042         \expandafter\@secondoftwo\bbl@eqdel}}
6043     \def\bbl@puteqno#1{\eqno\hbox{#1}}
6044     \def\bbl@putleqno#1{\leqno\hbox{#1}}
6045     \def\bbl@eqno@flip#1{%
6046       \ifdim\predisplaysize=-\maxdimen
6047         \eqno
6048         \hb@xt@.01pt{\hb@xt@\displaywidth{\hss{#1}}\hss}%
6049       \else
6050         \leqno\hbox{#1}%
6051       \fi}
6052     \def\bbl@leqno@flip#1{%
6053       \ifdim\predisplaysize=-\maxdimen
6054         \leqno
6055         \hb@xt@.01pt{\hss\hb@xt@\displaywidth{{#1}}\hss}%
6056       \else
6057         \eqno\hbox{#1}%
6058       \fi}
6059     \AtBeginDocument{%
6060       \ifx\maketag@@@\undefined % Normal equation, eqnarray
6061         \AddToHook{env/equation/begin}{%
6062           \ifnum\bbl@thetextdir>\z@
6063             \let\@eqnnum\bbl@eqnum
6064             \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6065             \chardef\bbl@thetextdir\z@
6066             \bbl@add\normalfont{\bbl@eqnodir}%
6067             \ifcase\bbl@eqnpos
6068               \let\bbl@puteqno\bbl@eqno@flip
6069             \or
6070               \let\bbl@puteqno\bbl@leqno@flip
6071             \fi
6072           \fi}%
6073       \ifnum\bbl@eqnpos=\tw@\else
6074         \def\endequation{\bbl@puteqno{\@eqnnum}$$\@ignoretrue}%
6075       \fi
6076       \AddToHook{env/eqnarray/begin}{%
6077         \ifnum\bbl@thetextdir>\z@

```

```

6078 \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6079 \chardef\bb1@thetextdir\z@
6080 \bb1@add\normalfont{\bb1@eqnodir}%
6081 \ifnum\bb1@eqnpos=\@ne
6082 \def\@eqnnum{%
6083 \setbox\z@\hbox{\bb1@eqnum}%
6084 \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6085 \else
6086 \let\@eqnnum\bb1@eqnum
6087 \fi
6088 \fi}
6089 % Hack. YA luatex bug?:
6090 \expandafter\bb1@sreplace\csname] \endcsname{${$}{\eqno\kern.001pt$}}%
6091 \else % amstex
6092 \ifx\bb1@noamsmath\undefined
6093 \bb1@exp{% Hack to hide maybe undefined conditionals:
6094 \chardef\bb1@eqnpos=0%
6095 \<iftagsleft@>1\<else>\<if@f1eqn>2\<fi>\<fi>\relax}%
6096 \ifnum\bb1@eqnpos=\@ne
6097 \let\bb1@ams@lap\hbox
6098 \else
6099 \let\bb1@ams@lap\llap
6100 \fi
6101 \ExplSyntaxOn
6102 \bb1@sreplace\intertext@{\normalbaselines}%
6103 {\normalbaselines
6104 \ifx\bb1@eqnodir\relax\else\bb1@pardir\@ne\bb1@eqnodir\fi}%
6105 \ExplSyntaxOff
6106 \def\bb1@ams@tagbox#1#2{#1{\bb1@eqnodir#2}}% #1=hbox|@lap|flip
6107 \ifx\bb1@ams@lap\hbox % leqno
6108 \def\bb1@ams@flip#1{%
6109 \hbox to 0.01pt{\hss\hbox to\displaywidth{{#1}\hss}}}%
6110 \else % eqno
6111 \def\bb1@ams@flip#1{%
6112 \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}\hss}}}%
6113 \fi
6114 \def\bb1@ams@preset#1{%
6115 \ifnum\bb1@thetextdir>\z@
6116 \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6117 \bb1@sreplace\textdef@{\hbox}{\bb1@ams@tagbox\hbox}%
6118 \bb1@sreplace\maketag@@@{\hbox}{\bb1@ams@tagbox#1}%
6119 \fi}%
6120 \ifnum\bb1@eqnpos=\tw@ \else
6121 \def\bb1@ams@equation{%
6122 \ifnum\bb1@thetextdir>\z@
6123 \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6124 \chardef\bb1@thetextdir\z@
6125 \bb1@add\normalfont{\bb1@eqnodir}%
6126 \ifcase\bb1@eqnpos
6127 \def\veqno##1##2{\bb1@eqno@flip{##1##2}}%
6128 \or
6129 \def\veqno##1##2{\bb1@leqno@flip{##1##2}}%
6130 \fi
6131 \fi}%
6132 \AddToHook{env/equation/begin}{\bb1@ams@equation}%
6133 \AddToHook{env/equation*/begin}{\bb1@ams@equation}%
6134 \fi
6135 \AddToHook{env/cases/begin}{\bb1@ams@preset\bb1@ams@lap}%
6136 \AddToHook{env/multline/begin}{\bb1@ams@preset\hbox}%
6137 \AddToHook{env/gather/begin}{\bb1@ams@preset\bb1@ams@lap}%
6138 \AddToHook{env/gather*/begin}{\bb1@ams@preset\bb1@ams@lap}%
6139 \AddToHook{env/align/begin}{\bb1@ams@preset\bb1@ams@lap}%
6140 \AddToHook{env/align*/begin}{\bb1@ams@preset\bb1@ams@lap}%

```



```

6141 \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6142 % Hackish, for proper alignment. Don't ask me why it works!:
6143 \bbl@exp{% Avoid a 'visible' conditional
6144   \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\tag*{}\<fi>}}%
6145 \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6146 \AddToHook{env/split/before}{%
6147   \ifnum\bbl@thetextdir>\z@
6148     \bbl@ifsamestring\@currentenv{equation}%
6149     {\ifx\bbl@ams@lap\hbox % leqno
6150       \def\bbl@ams@flip#1{%
6151         \hbox to 0.01pt{\hbox to\displaywidth{{#1}\hss}\hss}}%
6152       \else
6153         \def\bbl@ams@flip#1{%
6154           \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}}%
6155       \fi}%
6156     }%
6157   \fi}%
6158 \fi
6159 \fi}
6160 \fi
6161 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
6162 \ifnum\bbl@bidimode>\z@
6163   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6164     \bbl@exp{%
6165       \def\\bbl@insidemath{0}%
6166       \mathdir\the\bodydir
6167       #1% Once entered in math, set boxes to restore values
6168       \<ifmmode>%
6169         \everyvbox{%
6170           \the\everyvbox
6171           \bodydir\the\bodydir
6172           \mathdir\the\mathdir
6173           \everyhbox{\the\everyhbox}%
6174           \everyvbox{\the\everyvbox}}%
6175         \everyhbox{%
6176           \the\everyhbox
6177           \bodydir\the\bodydir
6178           \mathdir\the\mathdir
6179           \everyhbox{\the\everyhbox}%
6180           \everyvbox{\the\everyvbox}}%
6181       \<fi>}}%
6182   \def\@hangfrom#1{%
6183     \setbox\@tempboxa\hbox{{#1}}%
6184     \hangindent\wd\@tempboxa
6185     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6186       \shapemode\@ne
6187     \fi
6188     \noindent\box\@tempboxa}
6189 \fi
6190 \IfBabelLayout{tabular}
6191   {\let\bbl@OL@tabular\@tabular
6192     \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6193     \let\bbl@NL@tabular\@tabular
6194     \AtBeginDocument{%
6195       \ifx\bbl@NL@tabular\@tabular\else
6196         \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6197       \let\bbl@NL@tabular\@tabular
6198     \fi}}
6199   {}
6200 \IfBabelLayout{lists}
6201   {\let\bbl@OL@list\list
6202     \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
6203     \let\bbl@NL@list\list

```

```

6204 \def\bbl@listparshape#1#2#3{%
6205 \parshape #1 #2 #3 %
6206 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6207 \shapemode\tw@
6208 \fi}}
6209 {}
6210 \IfBabelLayout{graphics}
6211 {\let\bbl@pictresetdir\relax
6212 \def\bbl@pictsetdir#1{%
6213 \ifcase\bbl@thetextdir
6214 \let\bbl@pictresetdir\relax
6215 \else
6216 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6217 \or\textdir TLT
6218 \else\bodydir TLT \textdir TLT
6219 \fi
6220 % \text\par\dir required in pgf:
6221 \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6222 \fi}%
6223 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6224 \directlua{
6225 Babel.get_picture_dir = true
6226 Babel.picture_has_bidi = 0
6227 %
6228 function Babel.picture_dir (head)
6229 if not Babel.get_picture_dir then return head end
6230 if Babel.hlist_has_bidi(head) then
6231 Babel.picture_has_bidi = 1
6232 end
6233 return head
6234 end
6235 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6236 "Babel.picture_dir")
6237 }%
6238 \AtBeginDocument{%
6239 \def\LS@rot{%
6240 \setbox\@outputbox\vbox{%
6241 \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
6242 \long\def\put(#1,#2)#3{%
6243 \@killglue
6244 % Try:
6245 \ifx\bbl@pictresetdir\relax
6246 \def\bbl@tempc{0}%
6247 \else
6248 \directlua{
6249 Babel.get_picture_dir = true
6250 Babel.picture_has_bidi = 0
6251 }%
6252 \setbox\z@\hb@xt@\z@{%
6253 \@defaultunitsset\@tempdimc{#1}\unitlength
6254 \kern\@tempdimc
6255 #3\hss}% TODO: #3 executed twice (below). That's bad.
6256 \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6257 \fi
6258 % Do:
6259 \@defaultunitsset\@tempdimc{#2}\unitlength
6260 \raise\@tempdimc\hb@xt@\z@{%
6261 \@defaultunitsset\@tempdimc{#1}\unitlength
6262 \kern\@tempdimc
6263 {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
6264 \ignorespaces}%
6265 \MakeRobust\put}%
6266 \AtBeginDocument

```

```

6267 {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir\@gobble}%
6268 \ifx\pgfpicture\@undefined\else % TODO. Allow deactivate?
6269 \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6270 \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6271 \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6272 \fi
6273 \ifx\tikzpicture\@undefined\else
6274 \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\z@}%
6275 \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6276 \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
6277 \fi
6278 \ifx\tcolorbox\@undefined\else
6279 \def\tcb@drawing@env@begin{%
6280 \csname tcb@before@tcb@split@state\endcsname
6281 \bbl@pictsetdir\tw@
6282 \begin{\kv tcb@graphenv}%
6283 \tcb@bbdraw%
6284 \tcb@apply@graph@patches
6285 }%
6286 \def\tcb@drawing@env@end{%
6287 \end{\kv tcb@graphenv}%
6288 \bbl@pictresetdir
6289 \csname tcb@after@tcb@split@state\endcsname
6290 }%
6291 \fi
6292 }}
6293 {}

```

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.

```

6294 \IfBabelLayout{counters*}%
6295 {\bbl@add\bbl@opt@layout{.counters.}%
6296 \AddToHook{shipout/before}{%
6297 \let\bbl@tempa\babelsublr
6298 \let\babelsublr\@firstofone
6299 \let\bbl@save@thepage\thepage
6300 \protected@edef\thepage{\thepage}%
6301 \let\babelsublr\bbl@tempa}%
6302 \AddToHook{shipout/after}{%
6303 \let\thepage\bbl@save@thepage}}}%
6304 \IfBabelLayout{counters}%
6305 {\let\bbl@OL@@textsuperscript\@textsuperscript
6306 \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6307 \let\bbl@latinarabic=\@arabic
6308 \let\bbl@OL@@arabic\@arabic
6309 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
6310 \@ifpackagewith{babel}{bidi=default}%
6311 {\let\bbl@asciroman=\@roman
6312 \let\bbl@OL@@roman\@roman
6313 \def\@roman#1{\babelsublr{\ensureascii\bbl@asciroman#1}}}%
6314 \let\bbl@asciiRoman=\@Roman
6315 \let\bbl@OL@@roman\@Roman
6316 \def\@Roman#1{\babelsublr{\ensureascii\bbl@asciiRoman#1}}}%
6317 \let\bbl@OL@labelenumii\labelenumii
6318 \def\labelenumii{\theenumii}%
6319 \let\bbl@OL@p@enumiii\p@enumiii
6320 \def\p@enumiii{\p@enumii}\theenumii{}}}%
6321 <<Footnote changes>>
6322 \IfBabelLayout{footnotes}%
6323 {\let\bbl@OL@footnote\footnote
6324 \BabelFootnote\footnote\language\language}%
6325 \BabelFootnote\localfootnote\language\language}%

```

```

6326 \BabelFootnote\mainfootnote{\}\{\}\}
6327 {}

```

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.

```

6328 \IfBabelLayout{extras}%
6329 {\let\bbl@OL@underline\underline
6330 \bbl@sreplace\underline{\$ \@underline}\bbl@nextfake$ \@underline}%
6331 \let\bbl@OL@LaTeX2e\LaTeX2e
6332 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6333 \if b\expandafter\@car\@series\@nil\boldmath\fi
6334 \babelsublr}%
6335 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}
6336 {}
6337 \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.

```

6338 (*transforms)
6339 Babel.linebreaking.replacements = {}
6340 Babel.linebreaking.replacements[0] = {} -- pre
6341 Babel.linebreaking.replacements[1] = {} -- post
6342 Babel.linebreaking.replacements[2] = {} -- post-line WIP
6343
6344 -- Discretionaries contain strings as nodes
6345 function Babel.str_to_nodes(fn, matches, base)
6346   local n, head, last
6347   if fn == nil then return nil end
6348   for s in string.utfvalues(fn(matches)) do
6349     if base.id == 7 then
6350       base = base.replace
6351     end
6352     n = node.copy(base)
6353     n.char = s
6354     if not head then
6355       head = n
6356     else
6357       last.next = n
6358     end
6359     last = n
6360   end
6361   return head
6362 end
6363
6364 Babel.fetch_subtext = {}
6365
6366 Babel.ignore_pre_char = function(node)
6367   return (node.lang == Babel.nohyphenation)
6368 end
6369
6370 -- Merging both functions doesn't seem feasible, because there are too
6371 -- many differences.

```

```

6372 Babel.fetch_subtext[0] = function(head)
6373   local word_string = ''
6374   local word_nodes = {}
6375   local lang
6376   local item = head
6377   local inmath = false
6378
6379   while item do
6380
6381     if item.id == 11 then
6382       inmath = (item.subtype == 0)
6383     end
6384
6385     if inmath then
6386       -- pass
6387
6388     elseif item.id == 29 then
6389       local locale = node.get_attribute(item, Babel.attr_locale)
6390
6391       if lang == locale or lang == nil then
6392         lang = lang or locale
6393         if Babel.ignore_pre_char(item) then
6394           word_string = word_string .. Babel.us_char
6395         else
6396           word_string = word_string .. unicode.utf8.char(item.char)
6397         end
6398         word_nodes[#word_nodes+1] = item
6399       else
6400         break
6401       end
6402
6403     elseif item.id == 12 and item.subtype == 13 then
6404       word_string = word_string .. ' '
6405       word_nodes[#word_nodes+1] = item
6406
6407       -- Ignore leading unrecognized nodes, too.
6408     elseif word_string ~= '' then
6409       word_string = word_string .. Babel.us_char
6410       word_nodes[#word_nodes+1] = item -- Will be ignored
6411     end
6412
6413     item = item.next
6414   end
6415
6416   -- Here and above we remove some trailing chars but not the
6417   -- corresponding nodes. But they aren't accessed.
6418   if word_string:sub(-1) == ' ' then
6419     word_string = word_string:sub(1,-2)
6420   end
6421   word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6422   return word_string, word_nodes, item, lang
6423 end
6424
6425 Babel.fetch_subtext[1] = function(head)
6426   local word_string = ''
6427   local word_nodes = {}
6428   local lang
6429   local item = head
6430   local inmath = false
6431
6432   while item do
6433
6434     if item.id == 11 then

```

```

6435     inmath = (item.subtype == 0)
6436 end
6437
6438 if inmath then
6439     -- pass
6440
6441 elseif item.id == 29 then
6442     if item.lang == lang or lang == nil then
6443         if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6444             lang = lang or item.lang
6445             word_string = word_string .. unicode.utf8.char(item.char)
6446             word_nodes[#word_nodes+1] = item
6447         end
6448     else
6449         break
6450     end
6451
6452 elseif item.id == 7 and item.subtype == 2 then
6453     word_string = word_string .. '='
6454     word_nodes[#word_nodes+1] = item
6455
6456 elseif item.id == 7 and item.subtype == 3 then
6457     word_string = word_string .. '|'
6458     word_nodes[#word_nodes+1] = item
6459
6460 -- (1) Go to next word if nothing was found, and (2) implicitly
6461 -- remove leading USs.
6462 elseif word_string == '' then
6463     -- pass
6464
6465 -- This is the responsible for splitting by words.
6466 elseif (item.id == 12 and item.subtype == 13) then
6467     break
6468
6469 else
6470     word_string = word_string .. Babel.us_char
6471     word_nodes[#word_nodes+1] = item -- Will be ignored
6472 end
6473
6474 item = item.next
6475 end
6476
6477 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6478 return word_string, word_nodes, item, lang
6479 end
6480
6481 function Babel.pre_hyphenate_replace(head)
6482     Babel.hyphenate_replace(head, 0)
6483 end
6484
6485 function Babel.post_hyphenate_replace(head)
6486     Babel.hyphenate_replace(head, 1)
6487 end
6488
6489 Babel.us_char = string.char(31)
6490
6491 function Babel.hyphenate_replace(head, mode)
6492     local u = unicode.utf8
6493     local lbkr = Babel.linebreaking.replacements[mode]
6494     if mode == 2 then mode = 0 end -- WIP
6495
6496     local word_head = head
6497

```

```

6498 while true do -- for each subtext block
6499
6500     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6501
6502     if Babel.debug then
6503         print()
6504         print((mode == 0) and '@@@<' or '@@@>', w)
6505     end
6506
6507     if nw == nil and w == '' then break end
6508
6509     if not lang then goto next end
6510     if not lbkr[lang] then goto next end
6511
6512     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6513     -- loops are nested.
6514     for k=1, #lbkr[lang] do
6515         local p = lbkr[lang][k].pattern
6516         local r = lbkr[lang][k].replace
6517         local attr = lbkr[lang][k].attr or -1
6518
6519         if Babel.debug then
6520             print('*****', p, mode)
6521         end
6522
6523         -- This variable is set in some cases below to the first *byte*
6524         -- after the match, either as found by u.match (faster) or the
6525         -- computed position based on sc if w has changed.
6526         local last_match = 0
6527         local step = 0
6528
6529         -- For every match.
6530         while true do
6531             if Babel.debug then
6532                 print('====')
6533             end
6534             local new -- used when inserting and removing nodes
6535
6536             local matches = { u.match(w, p, last_match) }
6537
6538             if #matches < 2 then break end
6539
6540             -- Get and remove empty captures (with ())'s, which return a
6541             -- number with the position), and keep actual captures
6542             -- (from (...)), if any, in matches.
6543             local first = table.remove(matches, 1)
6544             local last = table.remove(matches, #matches)
6545             -- Non re-fetched substrings may contain \31, which separates
6546             -- subsubstrings.
6547             if string.find(w:sub(first, last-1), Babel.us_char) then break end
6548
6549             local save_last = last -- with A()BC()D, points to D
6550
6551             -- Fix offsets, from bytes to unicode. Explained above.
6552             first = u.len(w:sub(1, first-1)) + 1
6553             last = u.len(w:sub(1, last-1)) -- now last points to C
6554
6555             -- This loop stores in a small table the nodes
6556             -- corresponding to the pattern. Used by 'data' to provide a
6557             -- predictable behavior with 'insert' (w_nodes is modified on
6558             -- the fly), and also access to 'remove'd nodes.
6559             local sc = first-1 -- Used below, too
6560             local data_nodes = {}

```

```

6561
6562     local enabled = true
6563     for q = 1, last-first+1 do
6564         data_nodes[q] = w_nodes[sc+q]
6565         if enabled
6566             and attr > -1
6567             and not node.has_attribute(data_nodes[q], attr)
6568         then
6569             enabled = false
6570         end
6571     end
6572
6573     -- This loop traverses the matched substring and takes the
6574     -- corresponding action stored in the replacement list.
6575     -- sc = the position in substr nodes / string
6576     -- rc = the replacement table index
6577     local rc = 0
6578
6579     while rc < last-first+1 do -- for each replacement
6580         if Babel.debug then
6581             print('.....', rc + 1)
6582         end
6583         sc = sc + 1
6584         rc = rc + 1
6585
6586         if Babel.debug then
6587             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6588             local ss = ''
6589             for itt in node.traverse(head) do
6590                 if itt.id == 29 then
6591                     ss = ss .. unicode.utf8.char(itt.char)
6592                 else
6593                     ss = ss .. '{' .. itt.id .. '}'
6594                 end
6595             end
6596             print('*****', ss)
6597         end
6598
6599         local crep = r[rc]
6600         local item = w_nodes[sc]
6601         local item_base = item
6602         local placeholder = Babel.us_char
6603         local d
6604
6605         if crep and crep.data then
6606             item_base = data_nodes[crep.data]
6607         end
6608
6609         if crep then
6610             step = crep.step or 0
6611         end
6612
6613         if (not enabled) or (crep and next(crep) == nil) then -- = {}
6614             last_match = save_last -- Optimization
6615             goto next
6616         end
6617
6618         elseif crep == nil or crep.remove then
6619             node.remove(head, item)
6620             table.remove(w_nodes, sc)
6621             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6622             sc = sc - 1 -- Nothing has been inserted.
6623             last_match = utf8.offset(w, sc+1+step)

```



```

6624         goto next
6625
6626     elseif crep and crep.kashida then -- Experimental
6627         node.set_attribute(item,
6628             Babel.attr_kashida,
6629             crep.kashida)
6630         last_match = utf8.offset(w, sc+1+step)
6631         goto next
6632
6633     elseif crep and crep.string then
6634         local str = crep.string(matches)
6635         if str == '' then -- Gather with nil
6636             node.remove(head, item)
6637             table.remove(w_nodes, sc)
6638             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6639             sc = sc - 1 -- Nothing has been inserted.
6640         else
6641             local loop_first = true
6642             for s in string.utfvalues(str) do
6643                 d = node.copy(item_base)
6644                 d.char = s
6645                 if loop_first then
6646                     loop_first = false
6647                     head, new = node.insert_before(head, item, d)
6648                     if sc == 1 then
6649                         word_head = head
6650                     end
6651                     w_nodes[sc] = d
6652                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6653                 else
6654                     sc = sc + 1
6655                     head, new = node.insert_before(head, item, d)
6656                     table.insert(w_nodes, sc, new)
6657                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6658                 end
6659                 if Babel.debug then
6660                     print('.....', 'str')
6661                     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6662                 end
6663             end -- for
6664             node.remove(head, item)
6665         end -- if ''
6666         last_match = utf8.offset(w, sc+1+step)
6667         goto next
6668
6669     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6670         d = node.new(7, 0) -- (disc, discretionary)
6671         d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
6672         d.post = Babel.str_to_nodes(crep.post, matches, item_base)
6673         d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6674         d.attr = item_base.attr
6675         if crep.pre == nil then -- TeXbook p96
6676             d.penalty = crep.penalty or tex.hyphenpenalty
6677         else
6678             d.penalty = crep.penalty or tex.exhyphenpenalty
6679         end
6680         placeholder = '|'
6681         head, new = node.insert_before(head, item, d)
6682
6683     elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6684         -- ERROR
6685
6686     elseif crep and crep.penalty then

```

```

6687         d = node.new(14, 0)  -- (penalty, userpenalty)
6688         d.attr = item_base.attr
6689         d.penalty = crep.penalty
6690         head, new = node.insert_before(head, item, d)
6691
6692     elseif crep and crep.space then
6693         -- 655360 = 10 pt = 10 * 65536 sp
6694         d = node.new(12, 13)  -- (glue, spaceskip)
6695         local quad = font.getfont(item_base.font).size or 655360
6696         node.setglue(d, crep.space[1] * quad,
6697                      crep.space[2] * quad,
6698                      crep.space[3] * quad)
6699         if mode == 0 then
6700             placeholder = ' '
6701         end
6702         head, new = node.insert_before(head, item, d)
6703
6704     elseif crep and crep.spacefactor then
6705         d = node.new(12, 13)  -- (glue, spaceskip)
6706         local base_font = font.getfont(item_base.font)
6707         node.setglue(d,
6708                    crep.spacefactor[1] * base_font.parameters['space'],
6709                    crep.spacefactor[2] * base_font.parameters['space_stretch'],
6710                    crep.spacefactor[3] * base_font.parameters['space_shrink'])
6711         if mode == 0 then
6712             placeholder = ' '
6713         end
6714         head, new = node.insert_before(head, item, d)
6715
6716     elseif mode == 0 and crep and crep.space then
6717         -- ERROR
6718
6719     end  -- ie replacement cases
6720
6721     -- Shared by disc, space and penalty.
6722     if sc == 1 then
6723         word_head = head
6724     end
6725     if crep.insert then
6726         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6727         table.insert(w_nodes, sc, new)
6728         last = last + 1
6729     else
6730         w_nodes[sc] = d
6731         node.remove(head, item)
6732         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6733     end
6734
6735     last_match = utf8.offset(w, sc+1+step)
6736
6737     ::next::
6738
6739 end  -- for each replacement
6740
6741 if Babel.debug then
6742     print('....', '/')
6743     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6744 end
6745
6746 end  -- for match
6747
6748 end  -- for patterns
6749

```

```

6750     ::next::
6751     word_head = nw
6752 end -- for substring
6753 return head
6754 end
6755
6756 -- This table stores capture maps, numbered consecutively
6757 Babel.capture_maps = {}
6758
6759 -- The following functions belong to the next macro
6760 function Babel.capture_func(key, cap)
6761     local ret = "[" .. cap:gsub('{{[0-9]}}', "]]..m[%1]..[" .. "]"
6762     local cnt
6763     local u = unicode.utf8
6764     ret, cnt = ret:gsub('{{[0-9]}|(^|+)|(.-)}', Babel.capture_func_map)
6765     if cnt == 0 then
6766         ret = u.gsub(ret, '{{(%x%x%x%x+)}',
6767             function (n)
6768                 return u.char(tonumber(n, 16))
6769             end)
6770     end
6771     ret = ret:gsub("%[%]%.%", '')
6772     ret = ret:gsub("%.%[%]%", '')
6773     return key .. [[=function(m) return ]] .. ret .. [[ end]]
6774 end
6775
6776 function Babel.capt_map(from, mapno)
6777     return Babel.capture_maps[mapno][from] or from
6778 end
6779
6780 -- Handle the {n|abc|ABC} syntax in captures
6781 function Babel.capture_func_map(capno, from, to)
6782     local u = unicode.utf8
6783     from = u.gsub(from, '{{(%x%x%x%x+)}',
6784         function (n)
6785             return u.char(tonumber(n, 16))
6786         end)
6787     to = u.gsub(to, '{{(%x%x%x%x+)}',
6788         function (n)
6789             return u.char(tonumber(n, 16))
6790         end)
6791     local froms = {}
6792     for s in string.utfcharacters(from) do
6793         table.insert(froms, s)
6794     end
6795     local cnt = 1
6796     table.insert(Babel.capture_maps, {})
6797     local mlen = table.getn(Babel.capture_maps)
6798     for s in string.utfcharacters(to) do
6799         Babel.capture_maps[mlen][froms[cnt]] = s
6800         cnt = cnt + 1
6801     end
6802     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
6803         (mlen) .. ").. " .. "["
6804 end
6805
6806 -- Create/Extend reversed sorted list of kashida weights:
6807 function Babel.capture_kashida(key, wt)
6808     wt = tonumber(wt)
6809     if Babel.kashida_wts then
6810         for p, q in ipairs(Babel.kashida_wts) do
6811             if wt == q then
6812                 break

```

```

6813     elseif wt > q then
6814         table.insert(Babel.kashida_wts, p, wt)
6815         break
6816     elseif table.getn(Babel.kashida_wts) == p then
6817         table.insert(Babel.kashida_wts, wt)
6818     end
6819 end
6820 else
6821     Babel.kashida_wts = { wt }
6822 end
6823 return 'kashida = ' .. wt
6824 end
6825 </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.

```

6826 <*basic-r>
6827 Babel = Babel or {}
6828
6829 Babel.bidi_enabled = true
6830
6831 require('babel-data-bidi.lua')
6832
6833 local characters = Babel.characters
6834 local ranges = Babel.ranges
6835
6836 local DIR = node.id("dir")

```

```

6837
6838 local function dir_mark(head, from, to, outer)
6839   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6840   local d = node.new(DIR)
6841   d.dir = '+' .. dir
6842   node.insert_before(head, from, d)
6843   d = node.new(DIR)
6844   d.dir = '-' .. dir
6845   node.insert_after(head, to, d)
6846 end
6847
6848 function Babel.bidi(head, ispar)
6849   local first_n, last_n          -- first and last char with nums
6850   local last_es                  -- an auxiliary 'last' used with nums
6851   local first_d, last_d          -- first and last char in L/R block
6852   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):

```

6853   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6854   local strong_lr = (strong == 'l') and 'l' or 'r'
6855   local outer = strong
6856
6857   local new_dir = false
6858   local first_dir = false
6859   local inmath = false
6860
6861   local last_lr
6862
6863   local type_n = ''
6864
6865   for item in node.traverse(head) do
6866
6867     -- three cases: glyph, dir, otherwise
6868     if item.id == node.id'glyph'
6869       or (item.id == 7 and item.subtype == 2) then
6870
6871       local itemchar
6872       if item.id == 7 and item.subtype == 2 then
6873         itemchar = item.replace.char
6874       else
6875         itemchar = item.char
6876       end
6877       local chardata = characters[itemchar]
6878       dir = chardata and chardata.d or nil
6879       if not dir then
6880         for nn, et in ipairs(ranges) do
6881           if itemchar < et[1] then
6882             break
6883           elseif itemchar <= et[2] then
6884             dir = et[3]
6885             break
6886           end
6887         end
6888       end
6889       dir = dir or 'l'
6890       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).

```

6891     if new_dir then
6892         attr_dir = 0
6893         for at in node.traverse(item.attr) do
6894             if at.number == Babel.attr_dir then
6895                 attr_dir = at.value % 3
6896             end
6897         end
6898         if attr_dir == 1 then
6899             strong = 'r'
6900         elseif attr_dir == 2 then
6901             strong = 'al'
6902         else
6903             strong = 'l'
6904         end
6905         strong_lr = (strong == 'l') and 'l' or 'r'
6906         outer = strong_lr
6907         new_dir = false
6908     end
6909
6910     if dir == 'nsm' then dir = strong end -- W1

```

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

```

6911     dir_real = dir -- We need dir_real to set strong below
6912     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:

```

6913     if strong == 'al' then
6914         if dir == 'en' then dir = 'an' end -- W2
6915         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6916         strong_lr = 'r' -- W3
6917     end

```

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

```

6918     elseif item.id == node.id'dir' and not inmath then
6919         new_dir = true
6920         dir = nil
6921     elseif item.id == node.id'math' then
6922         inmath = (item.subtype == 0)
6923     else
6924         dir = nil -- Not a char
6925     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>.

```

6926     if dir == 'en' or dir == 'an' or dir == 'et' then
6927         if dir ~= 'et' then
6928             type_n = dir
6929         end
6930         first_n = first_n or item
6931         last_n = last_es or item
6932         last_es = nil
6933     elseif dir == 'es' and last_n then -- W3+W6
6934         last_es = item
6935     elseif dir == 'cs' then -- it's right - do nothing
6936     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6937         if strong_lr == 'r' and type_n ~= '' then
6938             dir_mark(head, first_n, last_n, 'r')
6939         elseif strong_lr == 'l' and first_d and type_n == 'an' then
6940             dir_mark(head, first_n, last_n, 'r')
6941         dir_mark(head, first_d, last_d, outer)

```

```

6942     first_d, last_d = nil, nil
6943     elseif strong_lr == 'l' and type_n ~= '' then
6944         last_d = last_n
6945     end
6946     type_n = ''
6947     first_n, last_n = nil, nil
6948 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:

```

6949     if dir == 'l' or dir == 'r' then
6950         if dir ~= outer then
6951             first_d = first_d or item
6952             last_d = item
6953         elseif first_d and dir ~= strong_lr then
6954             dir_mark(head, first_d, last_d, outer)
6955             first_d, last_d = nil, nil
6956         end
6957     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>, resp'tly, 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.

```

6958     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6959         item.char = characters[item.char] and
6960             characters[item.char].m or item.char
6961     elseif (dir or new_dir) and last_lr ~= item then
6962         local mir = outer .. strong_lr .. (dir or outer)
6963         if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6964             for ch in node.traverse(node.next(last_lr)) do
6965                 if ch == item then break end
6966                 if ch.id == node.id'glyph' and characters[ch.char] then
6967                     ch.char = characters[ch.char].m or ch.char
6968                 end
6969             end
6970         end
6971     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).

```

6972     if dir == 'l' or dir == 'r' then
6973         last_lr = item
6974         strong = dir_real          -- Don't search back - best save now
6975         strong_lr = (strong == 'l') and 'l' or 'r'
6976     elseif new_dir then
6977         last_lr = nil
6978     end
6979 end

```

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

```

6980     if last_lr and outer == 'r' then
6981         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6982             if characters[ch.char] then
6983                 ch.char = characters[ch.char].m or ch.char
6984             end
6985         end
6986     end
6987     if first_n then
6988         dir_mark(head, first_n, last_n, outer)
6989     end

```

```

6990 if first_d then
6991     dir_mark(head, first_d, last_d, outer)
6992 end

```

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

```

6993 return node.prev(head) or head
6994 end
6995 </basic-r>

```

And here the Lua code for bidi=basic:

```

6996 <*basic>
6997 Babel = Babel or {}
6998
6999 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
7000
7001 Babel.fontmap = Babel.fontmap or {}
7002 Babel.fontmap[0] = {}      -- l
7003 Babel.fontmap[1] = {}      -- r
7004 Babel.fontmap[2] = {}      -- al/an
7005
7006 Babel.bidi_enabled = true
7007 Babel.mirroring_enabled = true
7008
7009 require('babel-data-bidi.lua')
7010
7011 local characters = Babel.characters
7012 local ranges = Babel.ranges
7013
7014 local DIR = node.id('dir')
7015 local GLYPH = node.id('glyph')
7016
7017 local function insert_implicit(head, state, outer)
7018     local new_state = state
7019     if state.sim and state.eim and state.sim ~= state.eim then
7020         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
7021         local d = node.new(DIR)
7022         d.dir = '+' .. dir
7023         node.insert_before(head, state.sim, d)
7024         local d = node.new(DIR)
7025         d.dir = '-' .. dir
7026         node.insert_after(head, state.eim, d)
7027     end
7028     new_state.sim, new_state.eim = nil, nil
7029     return head, new_state
7030 end
7031
7032 local function insert_numeric(head, state)
7033     local new
7034     local new_state = state
7035     if state.san and state.ean and state.san ~= state.ean then
7036         local d = node.new(DIR)
7037         d.dir = '+TLT'
7038         _, new = node.insert_before(head, state.san, d)
7039         if state.san == state.sim then state.sim = new end
7040         local d = node.new(DIR)
7041         d.dir = '-TLT'
7042         _, new = node.insert_after(head, state.ean, d)
7043         if state.ean == state.eim then state.eim = new end
7044     end
7045     new_state.san, new_state.ean = nil, nil
7046     return head, new_state
7047 end
7048

```



```

7049 -- TODO - \hbox with an explicit dir can lead to wrong results
7050 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7051 -- was made to improve the situation, but the problem is the 3-dir
7052 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7053 -- well.
7054
7055 function Babel.bidi(head, ispar, hdir)
7056   local d    -- d is used mainly for computations in a loop
7057   local prev_d = ''
7058   local new_d = false
7059
7060   local nodes = {}
7061   local outer_first = nil
7062   local inmath = false
7063
7064   local glue_d = nil
7065   local glue_i = nil
7066
7067   local has_en = false
7068   local first_et = nil
7069
7070   local ATDIR = Babel.attr_dir
7071
7072   local save_outer
7073   local temp = node.get_attribute(head, ATDIR)
7074   if temp then
7075     temp = temp % 3
7076     save_outer = (temp == 0 and 'l') or
7077                  (temp == 1 and 'r') or
7078                  (temp == 2 and 'al')
7079   elseif ispar then -- Or error? Shouldn't happen
7080     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7081   else -- Or error? Shouldn't happen
7082     save_outer = ('TRT' == hdir) and 'r' or 'l'
7083   end
7084   -- when the callback is called, we are just _after_ the box,
7085   -- and the textdir is that of the surrounding text
7086   -- if not ispar and hdir ~= tex.textdir then
7087   --   save_outer = ('TRT' == hdir) and 'r' or 'l'
7088   -- end
7089   local outer = save_outer
7090   local last = outer
7091   -- 'al' is only taken into account in the first, current loop
7092   if save_outer == 'al' then save_outer = 'r' end
7093
7094   local fontmap = Babel.fontmap
7095
7096   for item in node.traverse(head) do
7097
7098     -- In what follows, #node is the last (previous) node, because the
7099     -- current one is not added until we start processing the neutrals.
7100
7101     -- three cases: glyph, dir, otherwise
7102     if item.id == GLYPH
7103        or (item.id == 7 and item.subtype == 2) then
7104
7105       local d_font = nil
7106       local item_r
7107       if item.id == 7 and item.subtype == 2 then
7108         item_r = item.replace -- automatic discs have just 1 glyph
7109       else
7110         item_r = item
7111       end

```

```

7112     local chardata = characters[item_r.char]
7113     d = chardata and chardata.d or nil
7114     if not d or d == 'nsm' then
7115         for nn, et in ipairs(ranges) do
7116             if item_r.char < et[1] then
7117                 break
7118             elseif item_r.char <= et[2] then
7119                 if not d then d = et[3]
7120                 elseif d == 'nsm' then d_font = et[3]
7121                 end
7122                 break
7123             end
7124         end
7125     end
7126     d = d or 'l'
7127
7128     -- A short 'pause' in bidi for mapfont
7129     d_font = d_font or d
7130     d_font = (d_font == 'l' and 0) or
7131             (d_font == 'nsm' and 0) or
7132             (d_font == 'r' and 1) or
7133             (d_font == 'al' and 2) or
7134             (d_font == 'an' and 2) or nil
7135     if d_font and fontmap and fontmap[d_font][item_r.font] then
7136         item_r.font = fontmap[d_font][item_r.font]
7137     end
7138
7139     if new_d then
7140         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7141         if inmath then
7142             attr_d = 0
7143         else
7144             attr_d = node.get_attribute(item, ATDIR)
7145             attr_d = attr_d % 3
7146         end
7147         if attr_d == 1 then
7148             outer_first = 'r'
7149             last = 'r'
7150         elseif attr_d == 2 then
7151             outer_first = 'r'
7152             last = 'al'
7153         else
7154             outer_first = 'l'
7155             last = 'l'
7156         end
7157         outer = last
7158         has_en = false
7159         first_et = nil
7160         new_d = false
7161     end
7162
7163     if glue_d then
7164         if (d == 'l' and 'l' or 'r') ~= glue_d then
7165             table.insert(nodes, {glue_i, 'on', nil})
7166         end
7167         glue_d = nil
7168         glue_i = nil
7169     end
7170
7171     elseif item.id == DIR then
7172         d = nil
7173         if head ~= item then new_d = true end
7174

```

```

7175 elseif item.id == node.id'glue' and item.subtype == 13 then
7176     glue_d = d
7177     glue_i = item
7178     d = nil
7179
7180 elseif item.id == node.id'math' then
7181     inmath = (item.subtype == 0)
7182
7183 else
7184     d = nil
7185 end
7186
7187 -- AL <= EN/ET/ES      -- W2 + W3 + W6
7188 if last == 'al' and d == 'en' then
7189     d = 'an'           -- W3
7190 elseif last == 'al' and (d == 'et' or d == 'es') then
7191     d = 'on'           -- W6
7192 end
7193
7194 -- EN + CS/ES + EN      -- W4
7195 if d == 'en' and #nodes >= 2 then
7196     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7197         and nodes[#nodes-1][2] == 'en' then
7198         nodes[#nodes][2] = 'en'
7199     end
7200 end
7201
7202 -- AN + CS + AN         -- W4 too, because uax9 mixes both cases
7203 if d == 'an' and #nodes >= 2 then
7204     if (nodes[#nodes][2] == 'cs')
7205         and nodes[#nodes-1][2] == 'an' then
7206         nodes[#nodes][2] = 'an'
7207     end
7208 end
7209
7210 -- ET/EN                -- W5 + W7->l / W6->on
7211 if d == 'et' then
7212     first_et = first_et or (#nodes + 1)
7213 elseif d == 'en' then
7214     has_en = true
7215     first_et = first_et or (#nodes + 1)
7216 elseif first_et then    -- d may be nil here !
7217     if has_en then
7218         if last == 'l' then
7219             temp = 'l'    -- W7
7220         else
7221             temp = 'en'   -- W5
7222         end
7223     else
7224         temp = 'on'      -- W6
7225     end
7226     for e = first_et, #nodes do
7227         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7228     end
7229     first_et = nil
7230     has_en = false
7231 end
7232
7233 -- Force mathdir in math if ON (currently works as expected only
7234 -- with 'l')
7235 if inmath and d == 'on' then
7236     d = ('TRT' == tex.mathdir) and 'r' or 'l'
7237 end

```

```

7238
7239   if d then
7240       if d == 'al' then
7241           d = 'r'
7242           last = 'al'
7243       elseif d == 'l' or d == 'r' then
7244           last = d
7245       end
7246       prev_d = d
7247       table.insert(nodes, {item, d, outer_first})
7248   end
7249
7250   outer_first = nil
7251
7252 end
7253
7254 -- TODO -- repeated here in case EN/ET is the last node. Find a
7255 -- better way of doing things:
7256 if first_et then      -- dir may be nil here !
7257     if has_en then
7258         if last == 'l' then
7259             temp = 'l'    -- W7
7260         else
7261             temp = 'en'   -- W5
7262         end
7263     else
7264         temp = 'on'       -- W6
7265     end
7266     for e = first_et, #nodes do
7267         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7268     end
7269 end
7270
7271 -- dummy node, to close things
7272 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7273
7274 ----- NEUTRAL -----
7275
7276 outer = save_outer
7277 last = outer
7278
7279 local first_on = nil
7280
7281 for q = 1, #nodes do
7282     local item
7283
7284     local outer_first = nodes[q][3]
7285     outer = outer_first or outer
7286     last = outer_first or last
7287
7288     local d = nodes[q][2]
7289     if d == 'an' or d == 'en' then d = 'r' end
7290     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7291
7292     if d == 'on' then
7293         first_on = first_on or q
7294     elseif first_on then
7295         if last == d then
7296             temp = d
7297         else
7298             temp = outer
7299         end
7300         for r = first_on, q - 1 do

```

```

7301     nodes[r][2] = temp
7302     item = nodes[r][1]    -- MIRRORING
7303     if Babel.mirroring_enabled and item.id == GLYPH
7304         and temp == 'r' and characters[item.char] then
7305         local font_mode = ''
7306         if item.font > 0 and font.fonts[item.font].properties then
7307             font_mode = font.fonts[item.font].properties.mode
7308         end
7309         if font_mode ~= 'harf' and font_mode ~= 'plug' then
7310             item.char = characters[item.char].m or item.char
7311         end
7312     end
7313 end
7314 first_on = nil
7315 end
7316
7317 if d == 'r' or d == 'l' then last = d end
7318 end
7319
7320 ----- IMPLICIT, REORDER -----
7321
7322 outer = save_outer
7323 last = outer
7324
7325 local state = {}
7326 state.has_r = false
7327
7328 for q = 1, #nodes do
7329
7330     local item = nodes[q][1]
7331
7332     outer = nodes[q][3] or outer
7333
7334     local d = nodes[q][2]
7335
7336     if d == 'nsm' then d = last end          -- W1
7337     if d == 'en' then d = 'an' end
7338     local isdir = (d == 'r' or d == 'l')
7339
7340     if outer == 'l' and d == 'an' then
7341         state.san = state.san or item
7342         state.ean = item
7343     elseif state.san then
7344         head, state = insert_numeric(head, state)
7345     end
7346
7347     if outer == 'l' then
7348         if d == 'an' or d == 'r' then      -- im -> implicit
7349             if d == 'r' then state.has_r = true end
7350             state.sim = state.sim or item
7351             state.eim = item
7352         elseif d == 'l' and state.sim and state.has_r then
7353             head, state = insert_implicit(head, state, outer)
7354         elseif d == 'l' then
7355             state.sim, state.eim, state.has_r = nil, nil, false
7356         end
7357     else
7358         if d == 'an' or d == 'l' then
7359             if nodes[q][3] then -- nil except after an explicit dir
7360                 state.sim = item -- so we move sim 'inside' the group
7361             else
7362                 state.sim = state.sim or item
7363             end
7364         end
7365     end
7366 end

```

```

7364         state.eim = item
7365     elseif d == 'r' and state.sim then
7366         head, state = insert_implicit(head, state, outer)
7367     elseif d == 'r' then
7368         state.sim, state.eim = nil, nil
7369     end
7370 end
7371
7372 if isdir then
7373     last = d          -- Don't search back - best save now
7374 elseif d == 'on' and state.san then
7375     state.san = state.san or item
7376     state.ean = item
7377 end
7378
7379 end
7380
7381 return node.prev(head) or head
7382 end
7383 </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.

```

7384 <*nil>
7385 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language]
7386 \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.

```

7387 \ifx\l@nil\@undefined
7388   \newlanguage\l@nil
7389   \@namedef{bbl@hyphendata@the\l@nil}{}}}% Remove warning
7390   \let\bbl@elt\relax
7391   \edef\bbl@languages{% Add it to the list of languages
7392     \bbl@languages\bbl@elt{nil}{the\l@nil}}}%
7393 \fi

```

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

```

7394 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

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

```

\captionnil
\datenil 7395 \let\captionnil\@empty
7396 \let\datenil\@empty

```

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

```

7397 \def\bbl@inidata@nil{%
7398   \bbl@elt{identification}{tag.ini}{und}%
7399   \bbl@elt{identification}{load.level}{0}%
7400   \bbl@elt{identification}{charset}{utf8}%
7401   \bbl@elt{identification}{version}{1.0}%
7402   \bbl@elt{identification}{date}{2022-05-16}%
7403   \bbl@elt{identification}{name.local}{nil}%
7404   \bbl@elt{identification}{name.english}{nil}%
7405   \bbl@elt{identification}{name.babel}{nil}%
7406   \bbl@elt{identification}{tag.bcp47}{und}%
7407   \bbl@elt{identification}{language.tag.bcp47}{und}%
7408   \bbl@elt{identification}{tag.opentype}{dflt}%
7409   \bbl@elt{identification}{script.name}{Latin}%
7410   \bbl@elt{identification}{script.tag.bcp47}{Latn}%
7411   \bbl@elt{identification}{script.tag.opentype}{DFLT}%
7412   \bbl@elt{identification}{level}{1}%
7413   \bbl@elt{identification}{encodings}{}%
7414   \bbl@elt{identification}{derivate}{no}}
7415 \@namedef{bbl@tbc@nil}{und}
7416 \@namedef{bbl@lbc@nil}{und}
7417 \@namedef{bbl@lotf@nil}{dflt}
7418 \@namedef{bbl@elname@nil}{nil}
7419 \@namedef{bbl@lname@nil}{nil}
7420 \@namedef{bbl@esname@nil}{Latin}
7421 \@namedef{bbl@sname@nil}{Latin}
7422 \@namedef{bbl@sbc@nil}{Latn}
7423 \@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.

```

7424 \ldf@finish{nil}
7425 \</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.

```

7426 \<Compute Julian day> ≡
7427 \def\bbl@fpmo#1#2{(#1-#2*floo(#1/#2))}
7428 \def\bbl@cs@gregleap#1{%
7429   (\bbl@fpmo{#1}{4} == 0) &&
7430   (!((\bbl@fpmo{#1}{100} == 0) && (\bbl@fpmo{#1}{400} != 0)))}
7431 \def\bbl@cs@jd#1#2#3{% year, month, day
7432   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
7433     floo((#1 - 1) / 4) + (-floo((#1 - 1) / 100)) +
7434     floo((#1 - 1) / 400) + floo((((367 * #2) - 362) / 12) +
7435     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3) }}
7436 \</Compute Julian day>

```

### 15.1 Islamic

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

```

7437 \<ca-islamic>
7438 \ExplSyntaxOn

```

```

7439 <<Compute Julian day>>
7440 % == islamic (default)
7441 % Not yet implemented
7442 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{
The Civil calendar.
7443 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
7444 ((#3 + ceil(29.5 * (#2 - 1)) +
7445 (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
7446 1948439.5) - 1) }
7447 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
7448 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
7449 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
7450 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
7451 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
7452 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
7453 \edef\bbl@tempa{%
7454 \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
7455 \edef#5{%
7456 \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
7457 \edef#6{\fp_eval:n{
7458 min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
7459 \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).
7460 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
7461 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
7462 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
7463 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
7464 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
7465 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
7466 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
7467 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
7468 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
7469 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
7470 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
7471 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
7472 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
7473 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
7474 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
7475 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
7476 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
7477 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
7478 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
7479 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
7480 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
7481 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
7482 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
7483 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
7484 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
7485 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
7486 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
7487 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
7488 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
7489 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
7490 65401,65431,65460,65490,65520}
7491 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
7492 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
7493 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
7494 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
7495 \ifnum#2>2014 \ifnum#2<2038

```



```

7496 \bbl@afterfi\expandafter\@gobble
7497 \fi\fi
7498 {\bbl@error{Year~out~of~range}{The~allowed~range~is~2014-2038}}%
7499 \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
7500 \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
7501 \count@\@ne
7502 \bbl@foreach\bbl@cs@umalqura@data{%
7503 \advance\count@\@ne
7504 \ifnum##1>\bbl@tempd\else
7505 \edef\bbl@tempe{\the\count@}%
7506 \edef\bbl@tempb{##1}%
7507 \fi}%
7508 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month-lunar
7509 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1) / 12) }}% annus
7510 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
7511 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
7512 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
7513 \ExplSyntaxOff
7514 \bbl@add\bbl@precalendar{%
7515 \bbl@replace\bbl@ld@calendar{-civil}}}%
7516 \bbl@replace\bbl@ld@calendar{-umalqura}}}%
7517 \bbl@replace\bbl@ld@calendar{+}}}%
7518 \bbl@replace\bbl@ld@calendar{-}}}%
7519 </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 hebcal.sty

```

7520 <*ca-hebrew>
7521 \newcount\bbl@cntcommon
7522 \def\bbl@remainder#1#2#3{%
7523 #3=#1\relax
7524 \divide #3 by #2\relax
7525 \multiply #3 by -#2\relax
7526 \advance #3 by #1\relax}%
7527 \newif\ifbbl@divisible
7528 \def\bbl@checkifdivisible#1#2{%
7529 {\countdef\tmp=0
7530 \bbl@remainder{#1}{#2}{\tmp}%
7531 \ifnum \tmp=0
7532 \global\bbl@divisibletrue
7533 \else
7534 \global\bbl@divisiblefalse
7535 \fi}}
7536 \newif\ifbbl@gregleap
7537 \def\bbl@ifgregleap#1{%
7538 \bbl@checkifdivisible{#1}{4}%
7539 \ifbbl@divisible
7540 \bbl@checkifdivisible{#1}{100}%
7541 \ifbbl@divisible
7542 \bbl@checkifdivisible{#1}{400}%
7543 \ifbbl@divisible
7544 \bbl@gregleaptrue
7545 \else
7546 \bbl@gregleapfalse
7547 \fi
7548 \else
7549 \bbl@gregleaptrue
7550 \fi
7551 \else

```

```

7552     \bbl@gregleapfalse
7553 \fi
7554 \ifbbl@gregleap}
7555 \def\bbl@gregdayspriormonths#1#2#3{%
7556     {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
7557         181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
7558     \bbl@ifgregleap{#2}%
7559     \ifnum #1 > 2
7560         \advance #3 by 1
7561     \fi
7562 \fi
7563 \global\bbl@cntcommon=#3}%
7564 #3=\bbl@cntcommon}
7565 \def\bbl@gregdaysprioryears#1#2{%
7566     {\countdef\tmpc=4
7567     \countdef\tmpb=2
7568     \tmpb=#1\relax
7569     \advance \tmpb by -1
7570     \tmpc=\tmpb
7571     \multiply \tmpc by 365
7572     #2=\tmpc
7573     \tmpc=\tmpb
7574     \divide \tmpc by 4
7575     \advance #2 by \tmpc
7576     \tmpc=\tmpb
7577     \divide \tmpc by 100
7578     \advance #2 by -\tmpc
7579     \tmpc=\tmpb
7580     \divide \tmpc by 400
7581     \advance #2 by \tmpc
7582     \global\bbl@cntcommon=#2\relax}%
7583 #2=\bbl@cntcommon}
7584 \def\bbl@absfromgreg#1#2#3#4{%
7585     {\countdef\tmpd=0
7586     #4=#1\relax
7587     \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
7588     \advance #4 by \tmpd
7589     \bbl@gregdaysprioryears{#3}{\tmpd}%
7590     \advance #4 by \tmpd
7591     \global\bbl@cntcommon=#4\relax}%
7592 #4=\bbl@cntcommon}
7593 \newif\ifbbl@hebrleap
7594 \def\bbl@checkleaphebryear#1{%
7595     {\countdef\tmpa=0
7596     \countdef\tmpb=1
7597     \tmpa=#1\relax
7598     \multiply \tmpa by 7
7599     \advance \tmpa by 1
7600     \bbl@remainder{\tmpa}{19}{\tmpb}%
7601     \ifnum \tmpb < 7
7602         \global\bbl@hebrleaptrue
7603     \else
7604         \global\bbl@hebrleapfalse
7605     \fi}}
7606 \def\bbl@hebreleapsedmonths#1#2{%
7607     {\countdef\tmpa=0
7608     \countdef\tmpb=1
7609     \countdef\tmpc=2
7610     \tmpa=#1\relax
7611     \advance \tmpa by -1
7612     #2=\tmpa
7613     \divide #2 by 19
7614     \multiply #2 by 235

```

```

7615 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
7616 \tmpc=\tmpb
7617 \multiply \tmpb by 12
7618 \advance #2 by \tmpb
7619 \multiply \tmpc by 7
7620 \advance \tmpc by 1
7621 \divide \tmpc by 19
7622 \advance #2 by \tmpc
7623 \global\bbl@cntcommon=#2}%
7624 #2=\bbl@cntcommon}
7625 \def\bbl@hebreleapseddays#1#2{%
7626 {\countdef\tmpa=0
7627 \countdef\tmpb=1
7628 \countdef\tmpc=2
7629 \bbl@hebreleapsedmonths{#1}{#2}%
7630 \tmpa=#2\relax
7631 \multiply \tmpa by 13753
7632 \advance \tmpa by 5604
7633 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
7634 \divide \tmpa by 25920
7635 \multiply #2 by 29
7636 \advance #2 by 1
7637 \advance #2 by \tmpa
7638 \bbl@remainder{#2}{7}{\tmpa}%
7639 \ifnum \tmpc < 19440
7640 \ifnum \tmpc < 9924
7641 \else
7642 \ifnum \tmpa=2
7643 \bbl@checkleaphebrewyear{#1}% of a common year
7644 \ifbbl@hebrleap
7645 \else
7646 \advance #2 by 1
7647 \fi
7648 \fi
7649 \fi
7650 \ifnum \tmpc < 16789
7651 \else
7652 \ifnum \tmpa=1
7653 \advance #1 by -1
7654 \bbl@checkleaphebrewyear{#1}% at the end of leap year
7655 \ifbbl@hebrleap
7656 \advance #2 by 1
7657 \fi
7658 \fi
7659 \fi
7660 \else
7661 \advance #2 by 1
7662 \fi
7663 \bbl@remainder{#2}{7}{\tmpa}%
7664 \ifnum \tmpa=0
7665 \advance #2 by 1
7666 \else
7667 \ifnum \tmpa=3
7668 \advance #2 by 1
7669 \else
7670 \ifnum \tmpa=5
7671 \advance #2 by 1
7672 \fi
7673 \fi
7674 \fi
7675 \global\bbl@cntcommon=#2\relax}%
7676 #2=\bbl@cntcommon}
7677 \def\bbl@daysinhebrewyear#1#2{%

```

```

7678 {\countdef\tmpe=12
7679 \bbl@hebrelapseddays{#1}{\tmpe}%
7680 \advance #1 by 1
7681 \bbl@hebrelapseddays{#1}{#2}%
7682 \advance #2 by -\tmpe
7683 \global\bbl@cntcommon=#2}%
7684 #2=\bbl@cntcommon}
7685 \def\bbl@hebrdayspriormonths#1#2#3{%
7686 {\countdef\tmpf= 14
7687 #3=\ifcase #1\relax
7688 0 \or
7689 0 \or
7690 30 \or
7691 59 \or
7692 89 \or
7693 118 \or
7694 148 \or
7695 148 \or
7696 177 \or
7697 207 \or
7698 236 \or
7699 266 \or
7700 295 \or
7701 325 \or
7702 400
7703 \fi
7704 \bbl@checkleaphebyear{#2}%
7705 \ifbbl@hebrleap
7706 \ifnum #1 > 6
7707 \advance #3 by 30
7708 \fi
7709 \fi
7710 \bbl@daysinhebyear{#2}{\tmpf}%
7711 \ifnum #1 > 3
7712 \ifnum \tmpf=353
7713 \advance #3 by -1
7714 \fi
7715 \ifnum \tmpf=383
7716 \advance #3 by -1
7717 \fi
7718 \fi
7719 \ifnum #1 > 2
7720 \ifnum \tmpf=355
7721 \advance #3 by 1
7722 \fi
7723 \ifnum \tmpf=385
7724 \advance #3 by 1
7725 \fi
7726 \fi
7727 \global\bbl@cntcommon=#3\relax}%
7728 #3=\bbl@cntcommon}
7729 \def\bbl@absfromhebr#1#2#3#4{%
7730 {#4=#1\relax
7731 \bbl@hebrdayspriormonths{#2}{#3}{#1}%
7732 \advance #4 by #1\relax
7733 \bbl@hebrelapseddays{#3}{#1}%
7734 \advance #4 by #1\relax
7735 \advance #4 by -1373429
7736 \global\bbl@cntcommon=#4\relax}%
7737 #4=\bbl@cntcommon}
7738 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
7739 {\countdef\tmpx= 17
7740 \countdef\tmpy= 18

```

```

7741 \countdef\tmpz= 19
7742 #6=#3\relax
7743 \global\advance #6 by 3761
7744 \bbl@absfromgreg{#1}{#2}{#3}{#4}%
7745 \tmpz=1 \tmpy=1
7746 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7747 \ifnum \tmpx > #4\relax
7748 \global\advance #6 by -1
7749 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7750 \fi
7751 \advance #4 by -\tmpx
7752 \advance #4 by 1
7753 #5=#4\relax
7754 \divide #5 by 30
7755 \loop
7756 \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
7757 \ifnum \tmpx < #4\relax
7758 \advance #5 by 1
7759 \tmpy=\tmpx
7760 \repeat
7761 \global\advance #5 by -1
7762 \global\advance #4 by -\tmpy}}
7763 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
7764 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
7765 \def\bbl@ca@hebrew#1-#2-#3\@#4#5#6{%
7766 \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
7767 \bbl@hebrfromgreg
7768 {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
7769 {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
7770 \edef#4{\the\bbl@hebryear}%
7771 \edef#5{\the\bbl@hebrmonth}%
7772 \edef#6{\the\bbl@hebrday}}
7773 </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 LPP 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).

```

7774 <*ca-persian>
7775 \ExplSyntaxOn
7776 <<Compute Julian day>>
7777 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
7778 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
7779 \def\bbl@ca@persian#1-#2-#3\@#4#5#6{%
7780 \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
7781 \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
7782 \bbl@afterfi\expandafter\@gobble
7783 \fi\fi
7784 {\bbl@error{Year~out~of~range}{The~allowed~range~is~2013-2050}}%
7785 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
7786 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
7787 \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
7788 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
7789 \ifnum\bbl@tempc<\bbl@tempb
7790 \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
7791 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
7792 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
7793 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
7794 \fi

```

```

7795 \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
7796 \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
7797 \edef#5{\fp_eval:n{% set Jalali month
7798   (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
7799 \edef#6{\fp_eval:n{% set Jalali day
7800   (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : ((#5 - 1) * 30) + 6))}}
7801 \ExplSyntaxOff
7802 </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.

```

7803 <*ca-coptic>
7804 \ExplSyntaxOn
7805 <<Compute Julian day>>
7806 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
7807   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
7808   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
7809   \edef#4{\fp_eval:n{%
7810     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
7811   \edef\bbl@tempc{\fp_eval:n{%
7812     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
7813   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
7814   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}%
7815 \ExplSyntaxOff
7816 </ca-coptic>
7817 <*ca-ethiopic>
7818 \ExplSyntaxOn
7819 <<Compute Julian day>>
7820 \def\bbl@ca@ethiopic#1-#2-#3\@#4#5#6{%
7821   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
7822   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
7823   \edef#4{\fp_eval:n{%
7824     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
7825   \edef\bbl@tempc{\fp_eval:n{%
7826     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
7827   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
7828   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}%
7829 \ExplSyntaxOff
7830 </ca-ethiopic>

```

## 19 Buddhist

That's very simple.

```

7831 <*ca-buddhist>
7832 \def\bbl@ca@buddhist#1-#2-#3\@#4#5#6{%
7833   \edef#4{\number\numexpr#1+543\relax}%
7834   \edef#5{#2}%
7835   \edef#6{#3}%
7836 </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:

People can have a file `locallyhyphen.tex` or whatever they like, but they mustn't diddle with `hyphen.tex` (or `plain.tex` except to preload additional fonts).

As these files are going to be read as the first thing  $\text{\LaTeX}$  sees, we need to set some category codes just to be able to change the definition of `\input`.

If a file called `hyphen.cfg` can be found, we make sure that *it* will be read instead of the file `hyphen.tex`. We do this by first saving the original meaning of `\input` (and I use a one letter control sequence for that so as not to waste multi-letter control sequence on this in the format).

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.

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

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

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

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).

214

```

7866 \immediate\write16{*}%
7867 }
7868 \input #1.cfg\relax
7869 \fi
7870 \@endofldf}

```

## 20.3 General tools

A number of  $\text{\LaTeX}$  macro's that are needed later on.

```

7871 \long\def\@firstofone#1{#1}
7872 \long\def\@firstoftwo#1#2{#1}
7873 \long\def\@secondoftwo#1#2{#2}
7874 \def\@nnil{\@nil}
7875 \def\@gobbletwo#1#2{}
7876 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7877 \def\@star@or@long#1{%
7878   \@ifstar
7879   {\let\@ngrel@x\relax#1}%
7880   {\let\@ngrel@x\long#1}}
7881 \let\@ngrel@x\relax
7882 \def\@car#1#2\@nil{#1}
7883 \def\@cdr#1#2\@nil{#2}
7884 \let\@typeset@protect\relax
7885 \let\protected@edef\edef
7886 \long\def\@gobble#1{}
7887 \edef\@backslashchar{\expandafter\@gobble\string\}
7888 \def\strip@prefix#1>{}
7889 \def\g@addto@macro#1#2{{%
7890   \toks@\expandafter{#1#2}%
7891   \xdef#1{\the\toks@}}}
7892 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
7893 \def\@nameuse#1{\csname #1\endcsname}
7894 \def\@ifundefined#1{%
7895   \expandafter\ifx\csname#1\endcsname\relax
7896   \expandafter\@firstoftwo
7897   \else
7898   \expandafter\@secondoftwo
7899   \fi}
7900 \def\@expandtwoargs#1#2#3{%
7901   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7902 \def\zap@space#1 #2{%
7903   #1%
7904   \ifx#2\@empty\else\expandafter\zap@space\fi
7905   #2}
7906 \let\bbl@trace\@gobble
7907 \def\bbl@error#1#2{%
7908   \begingroup
7909     \newlinechar=`^^J
7910     \def\{^^J(babel) }%
7911     \errhelp{#2}\errmessage{\#1}%
7912   \endgroup}
7913 \def\bbl@warning#1{%
7914   \begingroup
7915     \newlinechar=`^^J
7916     \def\{^^J(babel) }%
7917     \message{\#1}%
7918   \endgroup}
7919 \let\bbl@infowarn\bbl@warning
7920 \def\bbl@info#1{%
7921   \begingroup
7922     \newlinechar=`^^J
7923     \def\{^^J}%
7924     \wlog{#1}%

```



```
7925 \endgroup}
```

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

```
7926 \ifx\@preamblecmds\@undefined
7927   \def\@preamblecmds{}
7928 \fi
7929 \def\@onlypreamble#1{%
7930   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
7931     \@preamblecmds\do#1}}
7932 \@onlypreamble\@onlypreamble
```

Mimick  $\LaTeX$ 's `\AtBeginDocument`; for this to work the user needs to add `\begin{document}` to his file.

```
7933 \def\begin{document}{%
7934   \@begin{document}hook
7935   \global\let\@begin{document}hook\@undefined
7936   \def\do##1{\global\let##1\@undefined}%
7937   \@preamblecmds
7938   \global\let\do\noexpand}
7939 \ifx\@begin{document}hook\@undefined
7940   \def\@begin{document}hook{}
7941 \fi
7942 \@onlypreamble\@begin{document}hook
7943 \def\AtBeginDocument{\g@addto@macro\@begin{document}hook}
```

We also have to mimick  $\LaTeX$ 's `\AtEndOfPackage`. Our replacement macro is much simpler; it stores its argument in `\@endoflfd`.

```
7944 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
7945 \@onlypreamble\AtEndOfPackage
7946 \def\@endoflfd{}
7947 \@onlypreamble\@endoflfd
7948 \let\bbl@afterlang\@empty
7949 \chardef\bbl@opt@hyphenmap\z@
```

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

```
7950 \catcode`\&=\z@
7951 \ifx&\if@filesw\@undefined
7952   \expandafter\let\csname if@filesw\expandafter\endcsname
7953     \csname iffalse\endcsname
7954 \fi
7955 \catcode`\&=4
```

Mimick  $\LaTeX$ 's commands to define control sequences.

```
7956 \def\newcommand{\@star@or@long\new@command}
7957 \def\new@command#1{%
7958   \@testopt{\@newcommand#1}0}
7959 \def\@newcommand#1[#2]{%
7960   \@ifnextchar [{\@xargdef#1[#2]}%
7961     {\@argdef#1[#2]}}
7962 \long\def\@argdef#1[#2]#3{%
7963   \@yargdef#1\@ne{#2}{#3}}
7964 \long\def\@xargdef#1[#2][#3]#4{%
7965   \expandafter\def\expandafter#1\expandafter{%
7966     \expandafter\@protected@testopt\expandafter #1%
7967     \csname\string#1\expandafter\endcsname{#3}}}%
7968 \expandafter\@yargdef \csname\string#1\endcsname
7969   \tw@{#2}{#4}}
7970 \long\def\@yargdef#1#2#3{%
7971   \@tempcnta#3\relax
7972   \advance \@tempcnta \@ne
7973   \let\@hash@\relax
```

```

7974 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7975 \@tempcntb #2%
7976 \@whilenum\@tempcntb <\@tempcnta
7977 \do{%
7978 \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
7979 \advance\@tempcntb \@ne}%
7980 \let\@hash@##%
7981 \l@ngrel@x\expandafter\def\expandafter#1\reserved@a{
7982 \def\providecommand{\@star@or@long\provide@command}
7983 \def\provide@command#1{%
7984 \begingroup
7985 \escapechar\m@ne\edef\@gtempa{\string#1}}%
7986 \endgroup
7987 \expandafter\ifundefined\@gtempa
7988 {\def\reserved@a{\new@command#1}}%
7989 {\let\reserved@a\relax
7990 \def\reserved@a{\new@command\reserved@a}}%
7991 \reserved@a}%

7992 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7993 \def\declare@robustcommand#1{%
7994 \edef\reserved@a{\string#1}%
7995 \def\reserved@b{#1}%
7996 \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7997 \edef#1{%
7998 \ifx\reserved@a\reserved@b
7999 \noexpand\x@protect
8000 \noexpand#1%
8001 \fi
8002 \noexpand\protect
8003 \expandafter\noexpand\csname
8004 \expandafter\@gobble\string#1 \endcsname
8005 }%
8006 \expandafter\new@command\csname
8007 \expandafter\@gobble\string#1 \endcsname
8008 }
8009 \def\x@protect#1{%
8010 \ifx\protect\@typeset@protect\else
8011 \@x@protect#1%
8012 \fi
8013 }
8014 \catcode`\&=\z@ % Trick to hide conditionals
8015 \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`.

```

8016 \def\bbl@tempa{\csname newif\endcsname&fin@}
8017 \catcode`\&=4
8018 \ifx\in@\undefined
8019 \def\in@#1#2{%
8020 \def\in@@##1##2##3\in@@{%
8021 \ifx\in@##2\in@false\else\in@true\fi}%
8022 \in@@#2#1\in@\in@@}
8023 \else
8024 \let\bbl@tempa\empty
8025 \fi
8026 \bbl@tempa

```

$\text{\LaTeX}$  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  $\text{\TeX}$  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).

```
8027 \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.

```
8028 \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.

```
8029 \ifx\@tempcnta\undefined
8030   \csname newcount\endcsname\@tempcnta\relax
8031 \fi
8032 \ifx\@tempcntb\undefined
8033   \csname newcount\endcsname\@tempcntb\relax
8034 \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`).

```
8035 \ifx\bye\undefined
8036   \advance\count10 by -2\relax
8037 \fi
8038 \ifx\@ifnextchar\undefined
8039   \def\@ifnextchar#1#2#3{%
8040     \let\reserved@d=#1%
8041     \def\reserved@a{#2}\def\reserved@b{#3}%
8042     \futurelet\@let@token\@ifnch}
8043   \def\@ifnch{%
8044     \ifx\@let@token\sptoken
8045       \let\reserved@c\@xifnch
8046     \else
8047       \ifx\@let@token\reserved@d
8048         \let\reserved@c\reserved@a
8049       \else
8050         \let\reserved@c\reserved@b
8051       \fi
8052     \fi
8053     \reserved@c}
8054   \def\:{\let\@sptoken= } \: % this makes \sptoken a space token
8055   \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
8056 \fi
8057 \def\@testopt#1#2{%
8058   \@ifnextchar[#{#1}{#1[#2]}}
8059 \def\@protected@testopt#1{%
8060   \ifx\protect\@typeset@protect
8061     \expandafter\@testopt
8062   \else
8063     \@x@protect#1%
8064   \fi}
8065 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8066   #2\relax}\fi}
8067 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8068   \else\expandafter\@gobble\fi{#1}}
```

## 20.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain  $\TeX$  environment.

```
8069 \def\DeclareTextCommand{%
8070   \@dec@text@cmd\providecommand
8071 }
8072 \def\ProvideTextCommand{%
8073   \@dec@text@cmd\providecommand
8074 }
8075 \def\DeclareTextSymbol#1#2#3{%
```

```

8076 \@dec@text@cmd\chardef#1{#2}#3\relax
8077 }
8078 \def\@dec@text@cmd#1#2#3{%
8079 \expandafter\def\expandafter#2%
8080 \expandafter{%
8081 \csname#3-cmd\expandafter\endcsname
8082 \expandafter#2%
8083 \csname#3\string#2\endcsname
8084 }%
8085 % \let\@ifdefinable\@rc@ifdefinable
8086 \expandafter#1\csname#3\string#2\endcsname
8087 }
8088 \def\@current@cmd#1{%
8089 \ifx\protect\@typeset@protect\else
8090 \noexpand#1\expandafter\@gobble
8091 \fi
8092 }
8093 \def\@changed@cmd#1#2{%
8094 \ifx\protect\@typeset@protect
8095 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8096 \expandafter\ifx\csname ?\string#1\endcsname\relax
8097 \expandafter\def\csname ?\string#1\endcsname{%
8098 \@changed@x@err{#1}%
8099 }%
8100 \fi
8101 \global\expandafter\let
8102 \csname\cf@encoding\string#1\expandafter\endcsname
8103 \csname ?\string#1\endcsname
8104 \fi
8105 \csname\cf@encoding\string#1%
8106 \expandafter\endcsname
8107 \else
8108 \noexpand#1%
8109 \fi
8110 }
8111 \def\@changed@x@err#1{%
8112 \errhelp{Your command will be ignored, type <return> to proceed}%
8113 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8114 \def\DeclareTextCommandDefault#1{%
8115 \DeclareTextCommand#1?%
8116 }
8117 \def\ProvideTextCommandDefault#1{%
8118 \ProvideTextCommand#1?%
8119 }
8120 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8121 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8122 \def\DeclareTextAccent#1#2#3{%
8123 \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
8124 }
8125 \def\DeclareTextCompositeCommand#1#2#3#4{%
8126 \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8127 \edef\reserved@b{\string##1}%
8128 \edef\reserved@c{%
8129 \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8130 \ifx\reserved@b\reserved@c
8131 \expandafter\expandafter\expandafter\ifx
8132 \expandafter\@car\reserved@a\relax\relax\@nil
8133 \@text@composite
8134 \else
8135 \edef\reserved@b##1{%
8136 \def\expandafter\noexpand
8137 \csname#2\string#1\endcsname###1{%
8138 \noexpand\@text@composite

```

```

8139         \expandafter\noexpand\csname#2\string#1\endcsname
8140         ###1\noexpand\@empty\noexpand\@text@composite
8141         {##1}%
8142     }%
8143 }%
8144     \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8145 \fi
8146     \expandafter\def\csname\expandafter\string\csname
8147     #2\endcsname\string#1-\string#3\endcsname{#4}
8148 \else
8149     \errhelp{Your command will be ignored, type <return> to proceed}%
8150     \errmessage{\string\DeclareTextCompositeCommand\space used on
8151     inappropriate command \protect#1}
8152 \fi
8153 }
8154 \def\@text@composite#1#2#3\@text@composite{%
8155     \expandafter\@text@composite@x
8156     \csname\string#1-\string#2\endcsname
8157 }
8158 \def\@text@composite@x#1#2{%
8159     \ifx#1\relax
8160         #2%
8161     \else
8162         #1%
8163     \fi
8164 }
8165 %
8166 \def\@strip@args#1:#2-#3\@strip@args{#2}
8167 \def\DeclareTextComposite#1#2#3#4{%
8168     \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8169     \bgroup
8170     \lccode`\@=#4%
8171     \lowercase{%
8172     \egroup
8173     \reserved@a @%
8174     }%
8175 }
8176 %
8177 \def\UseTextSymbol#1#2{#2}
8178 \def\UseTextAccent#1#2#3{}
8179 \def\@use@text@encoding#1{}
8180 \def\DeclareTextSymbolDefault#1#2{%
8181     \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8182 }
8183 \def\DeclareTextAccentDefault#1#2{%
8184     \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8185 }
8186 \def\cf@encoding{OT1}

```

Currently we only use the  $\text{\TeX}_{2\epsilon}$  method for accents for those that are known to be made active in *some* language definition file.

```

8187 \DeclareTextAccent{"}{OT1}{127}
8188 \DeclareTextAccent{'}{OT1}{19}
8189 \DeclareTextAccent{^}{OT1}{94}
8190 \DeclareTextAccent`}{OT1}{18}
8191 \DeclareTextAccent{~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for PLAIN  $\text{\TeX}$ .

```

8192 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8193 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
8194 \DeclareTextSymbol{\textquoteleft}{OT1}{``'}
8195 \DeclareTextSymbol{\textquoteright}{OT1}{``'}
8196 \DeclareTextSymbol{\i}{OT1}{16}
8197 \DeclareTextSymbol{\ss}{OT1}{25}

```

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

```
8198 \ifx\scriptsize\@undefined
8199   \let\scriptsize\sevenrm
8200 \fi
```

And a few more “dummy” definitions.

```
8201 \def\language{english}%
8202 \let\bbl@opt@shorthands\@nnil
8203 \def\bbl@ifshorthand#1#2#3{#2}%
8204 \let\bbl@language@opts\@empty
8205 \ifx\babeloptionstrings\@undefined
8206   \let\bbl@opt@strings\@nnil
8207 \else
8208   \let\bbl@opt@strings\babeloptionstrings
8209 \fi
8210 \def\BabelStringsDefault{generic}
8211 \def\bbl@tempa{normal}
8212 \ifx\babeloptionmath\bbl@tempa
8213   \def\bbl@mathnormal{\noexpand\textnormal}
8214 \fi
8215 \def\AfterBabelLanguage#1#2{}
8216 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
8217 \let\bbl@afterlang\relax
8218 \def\bbl@opt@safe{BR}
8219 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
8220 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
8221 \expandafter\newif\csname ifbbl@single\endcsname
8222 \chardef\bbl@bidimode\z@
8223 \</Emulate LaTeX>
```

A proxy file:

```
8224 \<plain>
8225 \input babel.def
8226 \</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  $\TeX$  styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The  $\TeX$ book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport,  *$\TeX$ , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in:  $\TeX$ 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 T<sub>E</sub>X*, *TUGboat* 9 (1988) #1, p. 70–72.
- [11] Joachim Schrod, *International E<sub>T</sub>X is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using E<sub>T</sub>X*, Springer, 2002, p. 301–373.
- [13] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).