

# 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 3.84.2989  
2023/01/21

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 . . . . .	11
1.10	Shorthands . . . . .	12
1.11	Package options . . . . .	15
1.12	The base option . . . . .	17
1.13	ini files . . . . .	18
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 . . . . .	46
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 . . . . .	51
<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 . . . . .	54
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 . . . . .	57
3.8	Encoding-dependent strings . . . . .	57
3.9	Executing code based on the selector . . . . .	60
<b>II</b>	<b>Source code</b>	<b>61</b>
<b>4</b>	<b>Identification and loading of required files</b>	<b>61</b>
<b>5</b>	<b>locale directory</b>	<b>61</b>

<b>6</b>	<b>Tools</b>	<b>62</b>
6.1	Multiple languages	66
6.2	The Package File ( <code>\LaTeX</code> , <code>babel.sty</code> )	66
6.3	<code>base</code>	68
6.4	<code>key=value</code> options and other general option	68
6.5	Conditional loading of shorthands	70
6.6	Interlude for Plain	71
<b>7</b>	<b>Multiple languages</b>	<b>71</b>
7.1	Selecting the language	74
7.2	Errors	82
7.3	Hooks	84
7.4	Setting up language files	86
7.5	Shorthands	88
7.6	Language attributes	96
7.7	Support for saving macro definitions	98
7.8	Short tags	99
7.9	Hyphens	100
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	108
7.12.2	Letters	109
7.12.3	Shorthands for quotation marks	110
7.12.4	Umlauts and tremas	111
7.13	Layout	112
7.14	Load engine specific macros	112
7.15	Creating and modifying languages	113
<b>8</b>	<b>Adjusting the Babel bahavior</b>	<b>134</b>
8.1	Cross referencing macros	136
8.2	Marks	139
8.3	Preventing clashes with other packages	140
8.3.1	<code>ifthen</code>	140
8.3.2	<code>varioref</code>	141
8.3.3	<code>hhline</code>	141
8.4	Encoding and fonts	142
8.5	Basic bidi support	143
8.6	Local Language Configuration	146
8.7	Language options	147
<b>9</b>	<b>The kernel of Babel (<code>babel.def</code>, <code>common</code>)</b>	<b>150</b>
<b>10</b>	<b>Loading hyphenation patterns</b>	<b>150</b>
<b>11</b>	<b>Font handling with <code>fontspec</code></b>	<b>154</b>
<b>12</b>	<b>Hooks for XeTeX and LuaTeX</b>	<b>158</b>
12.1	XeTeX	158
12.2	Layout	160
12.3	8-bit TeX	161
12.4	LuaTeX	162
12.5	Southeast Asian scripts	168
12.6	CJK line breaking	169
12.7	Arabic justification	172
12.8	Common stuff	175
12.9	Automatic fonts and ids switching	175
12.10	Bidi	181
12.11	Layout	184

12.12	Lua: transforms . . . . .	190
12.13	Lua: Auto bidi with basic and basic-r . . . . .	198
<b>13</b>	<b>Data for CJK</b>	<b>208</b>
<b>14</b>	<b>The ‘nil’ language</b>	<b>209</b>
<b>15</b>	<b>Calendars</b>	<b>210</b>
15.1	Islamic . . . . .	210
<b>16</b>	<b>Hebrew</b>	<b>211</b>
<b>17</b>	<b>Persian</b>	<b>215</b>
<b>18</b>	<b>Coptic and Ethiopic</b>	<b>216</b>
<b>19</b>	<b>Buddhist</b>	<b>217</b>
<b>20</b>	<b>Support for Plain T<sub>E</sub>X (plain.def)</b>	<b>217</b>
20.1	Not renaming hyphen.tex . . . . .	217
20.2	Emulating some L <sup>A</sup> T <sub>E</sub> X features . . . . .	218
20.3	General tools . . . . .	218
20.4	Encoding related macros . . . . .	222
<b>21</b>	<b>Acknowledgements</b>	<b>225</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’ . . . . .	9
Argument of \language@active@arg” has an extra } . . . . .	12
Package babel Info: The following fonts are not babel standard families . . . . .	27

# 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, TeXLive, 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.

**NOTE** Babel does not make any readjustments by default in font size, vertical positioning or line height by default. This is on purpose because the optimal solution depends on the document layout and the font, and very likely the most appropriate one is a combination of these settings.

## 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  $\LaTeX$ , the preamble of the document:

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

would tell  $\LaTeX$  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 xet<sub>ex</sub> and lua<sub>tex</sub>, 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:

LUATEX/XETEX

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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

**New 3.84** With `pdfTeX`, when a language is loaded on the fly (actually, with `\babelprovide`) selectors now set the font encoding based on the list provided when loading `fontenc`. Not all scripts have an associated encoding, so this feature works only with Latin, Cyrillic, Greek, Arabic, Hebrew, Cherokee, Armenian, and Georgian, provided a suitable font is found.

## 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}.
```

<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.

- 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.

## 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.

<sup>3</sup>In old versions the error read “You haven’t loaded the language LANG yet”.

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

- `\selectlanguage` should not be used inside some boxed environments (like floats or minipage) to switch the language if you need the information written to the aux be correctly synchronized. This rarely happens, but if it were the case, you must use other language 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 `bidir` 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  $\TeX$  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}`).

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

## 1.10 Shorthands

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

---

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

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).

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

The command `\usesshorthands` 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 `\usesshorthands*{\langle char \rangle}` is provided, which makes sure shorthands are always activated.

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

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

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

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

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

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

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

```
\usesshorthands*{"}
\defineshorthand{"*}{\babelhyphen{soft}}
\defineshorthand{"-}{\babelhyphen{hard}}
```

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

```
\defineshorthand[*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.

## `\languageshorthands` $\{\langle language \rangle\}$

The command `\languageshorthands` can be used to switch the shorthands on the language level. It takes one argument, the name of a language or none (the latter does what its name suggests).<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{\languageshorthands{ngerman}}
```

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

**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]{\{\languageshorthands{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** " ^ =

<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.

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

**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** {<character>}{<true>}{<false>}

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

**\aliasshorthand** {<original>}{<alias>}

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

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



**safe=** none | ref | bib

Some L<sup>A</sup>T<sub>E</sub>X 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  $\epsilon$ T<sub>E</sub>X 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 spoiled 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 other language;

<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.

**other\*** also sets it at `other language*` 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=** \*

**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** `{<option-name>}{<code>}`

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

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

does ... at the end of `french.ldf`. It can be used in `ldf` files, too, but in such a case the code is executed only if `<option-name>` 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 `TeX`, an alternative method to execute some code just after an `ldf` file is loaded is with `\AddToHook` and the hook `file/<language>.ldf/after`. `Babel` does not predeclare it, and you have to do it yourself with `\ActivateGenericHook`.

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

<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.

### 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 T<sub>E</sub>X and other systems, they are identified with the BCP 47 codes as preferred by the Unicode Common Locale Data Repository, which was used as source for most of the data provided by these files, too (the main exception being the \...name strings).

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

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

LUATEX/XETEX

```
\documentclass{book}

\usepackage{babel}
\babelprovide[import, main]{georgian}

\babelfont{rm}[Renderer=Harfbuzz]{DejaVu Sans}

\begin{document}

\tableofcontents

\chapter{სამშარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამშარეულო 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 luatex also applies here. Some quick patterns can help, with something similar to:

```
\babelprovide[import, hyphenrules=+]{lao}
\babelpatterns[lao]{lᦺ lᦴ lᦶ lᦸ lᦺ lᦴ lᦶ lᦸ} % 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>	ar-IQ	Arabic <sup>u</sup>
agq	Aghem	ar-JO	Arabic <sup>u</sup>
ak	Akan	ar-LB	Arabic <sup>u</sup>
am	Amharic <sup>ul</sup>	ar-MA	Arabic <sup>u</sup>
ar-DZ	Arabic <sup>u</sup>	ar-PS	Arabic <sup>u</sup>
ar-EG	Arabic <sup>u</sup>	ar-SA	Arabic <sup>u</sup>

ar-SY	Arabic <sup>u</sup>	en-NZ	English <sup>ul</sup>
ar-TN	Arabic <sup>u</sup>	en-US	American English <sup>ul</sup>
ar	Arabic <sup>u</sup>	en	English <sup>ul</sup>
as	Assamese <sup>u</sup>	eo	Esperanto <sup>ul</sup>
asa	Asu	es-MX	Mexican Spanish <sup>ul</sup>
ast	Asturian <sup>ul</sup>	es	Spanish <sup>ul</sup>
az-Cyrl	Azerbaijani	et	Estonian <sup>ul</sup>
az-Latn	Azerbaijani	eu	Basque <sup>ul</sup>
az	Azerbaijani <sup>ul</sup>	ewo	Ewondo
bas	Basaa	fa	Persian <sup>u</sup>
be	Belarusian <sup>ul</sup>	ff	Fulah
bem	Bemba	fi	Finnish <sup>ul</sup>
bez	Bena	fil	Filipino
bg	Bulgarian <sup>ul</sup>	fo	Faroese
bm	Bambara	fr-BE	French <sup>ul</sup>
bn	Bangla <sup>u</sup>	fr-CA	Canadian French <sup>ul</sup>
bo	Tibetan <sup>u</sup>	fr-CH	Swiss French <sup>ul</sup>
br	Breton <sup>ul</sup>	fr-LU	French <sup>ul</sup>
brx	Bodo	fr	French <sup>ul</sup>
bs-Cyrl	Bosnian	fur	Friulian <sup>ul</sup>
bs-Latn	Bosnian <sup>ul</sup>	fy	Western Frisian
bs	Bosnian <sup>ul</sup>	ga	Irish <sup>ul</sup>
ca	Catalan <sup>ul</sup>	gd	Scottish Gaelic <sup>ul</sup>
ce	Chechen	gl	Galician <sup>ul</sup>
cgg	Chiga	grc	Ancient Greek <sup>ul</sup>
chr	Cherokee	gsw	Swiss German
ckb-Arab	Central Kurdish <sup>u</sup>	gu	Gujarati
ckb-Latn	Central Kurdish <sup>u</sup>	guz	Gusii
ckb	Central Kurdish <sup>u</sup>	gv	Manx
cop	Coptic	ha-GH	Hausa
cs	Czech <sup>ul</sup>	ha-NE	Hausa
cu-Cyrs	Church Slavic <sup>u</sup>	ha	Hausa <sup>ul</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

kl	Kalaallisut	nus	Nuer
klh	Kalenjin	nyn	Nyankole
km	Khmer <sup>u</sup>	oc	Occitan <sup>ul</sup>
kmr-Arab	Northern Kurdish <sup>u</sup>	om	Oromo
kmr-Latn	Northern Kurdish <sup>ul</sup>	or	Odia
kmr	Northern Kurdish <sup>ul</sup>	os	Ossetic
kn	Kannada <sup>u</sup>	pa-Arab	Punjabi
ko-Hani	Korean <sup>u</sup>	pa-Guru	Punjabi <sup>u</sup>
ko	Korean <sup>u</sup>	pa	Punjabi <sup>u</sup>
kok	Konkani	pl	Polish <sup>ul</sup>
ks	Kashmiri	pms	Piedmontese <sup>ul</sup>
ksb	Shambala	ps	Pashto
ksf	Bafia	pt-BR	Brazilian Portuguese <sup>ul</sup>
ksh	Colognian	pt-PT	European Portuguese <sup>ul</sup>
kw	Cornish	pt	Portuguese <sup>ul</sup>
ky	Kyrgyz	qu	Quechua
la-x-classic	Classic Latin <sup>ul</sup>	rm	Romansh <sup>ul</sup>
la-x-ecclesia	Ecclesiastic Latin <sup>ul</sup>	rn	Rundi
la-x-medieval	Medieval Latin <sup>ul</sup>	ro-MD	Moldavian <sup>ul</sup>
la	Latin <sup>ul</sup>	ro	Romanian <sup>ul</sup>
lag	Langi	rof	Rombo
lb	Luxembourgish <sup>ul</sup>	ru	Russian <sup>ul</sup>
lg	Ganda	rw	Kinyarwanda
lkt	Lakota	rwk	Rwa
ln	Lingala	sa-Beng	Sanskrit
lo	Lao <sup>u</sup>	sa-Deva	Sanskrit
lrc	Northern Luri	sa-Gujr	Sanskrit
lt	Lithuanian <sup>ul</sup>	sa-Knda	Sanskrit
lu	Luba-Katanga	sa-Mlym	Sanskrit
luo	Luo	sa-Telu	Sanskrit
luy	Luyia	sa	Sanskrit
lv	Latvian <sup>ul</sup>	sah	Sakha
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>

sv	Swedish <sup>ul</sup>	vai	Vai
sw	Swahili	vi	Vietnamese <sup>ul</sup>
syr	Syriac	vun	Vunjo
ta	Tamil <sup>u</sup>	wae	Walser
te	Telugu <sup>u</sup>	xog	Soga
teo	Teso	yav	Yangben
th	Thai <sup>ul</sup>	yi	Yiddish
ti	Tigrinya	yo	Yoruba
tk	Turkmen <sup>ul</sup>	yrl	Nheengatu
to	Tongan	yue	Cantonese
tr	Turkish <sup>ul</sup>	zgh	Standard Moroccan Tamazight
twq	Tasawaq	zh-Hans-HK	Chinese
tzm	Central Atlas Tamazight	zh-Hans-MO	Chinese
ug	Uyghur <sup>u</sup>	zh-Hans-SG	Chinese
uk	Ukrainian <sup>ul</sup>	zh-Hans	Chinese <sup>u</sup>
ur	Urdu <sup>u</sup>	zh-Hant-HK	Chinese
uz-Arab	Uzbek	zh-Hant-MO	Chinese
uz-Cyrl	Uzbek	zh-Hant	Chinese <sup>u</sup>
uz-Latn	Uzbek	zh	Chinese <sup>u</sup>
uz	Uzbek	zu	Zulu
vai-Latn	Vai		
vai-Vaii	Vai		

---

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

---

afrikaans	basaa
aghem	basque
akan	belarusian
albanian	bemba
american	bena
amharic	bangla
ancientgreek	bodo
arabic	bosnian-cyrillic
arabic-algeria	bosnian-cyrl
arabic-DZ	bosnian-latin
arabic-morocco	bosnian-latn
arabic-MA	bosnian
arabic-syria	brazilian
arabic-SY	breton
armenian	british
assamese	bulgarian
asturian	burmese
asu	canadian
australian	cantonese
austrian	catalan
azerbaijani-cyrillic	centralatlastamazight
azerbaijani-cyrl	centralkurdish
azerbaijani-latin	chechen
azerbaijani-latn	cherokee
azerbaijani	chiga
bafia	chinese-hans-hk
bambara	chinese-hans-mo

chinese-hans-sg	galician
chinese-hans	ganda
chinese-hant-hk	georgian
chinese-hant-mo	german-at
chinese-hant	german-austria
chinese-simplified-hongkongsarchina	german-ch
chinese-simplified-macausarchina	german-switzerland
chinese-simplified-singapore	german
chinese-simplified	greek
chinese-traditional-hongkongsarchina	gujarati
chinese-traditional-macausarchina	gusii
chinese-traditional	hausa-gh
chinese	hausa-ghana
churchslavic	hausa-ne
churchslavic-cyrs	hausa-niger
churchslavic-oldcyrillic <sup>12</sup>	hausa
churchslavic-glag	hawaiian
churchslavic-glagolitic	hebrew
cognian	hindi
cornish	hungarian
croatian	icelandic
czech	igbo
danish	inarisami
duala	indonesian
dutch	interlingua
dzongkha	irish
embu	italian
english-au	japanese
english-australia	jolafonyi
english-ca	kabuverdianu
english-canada	kabyle
english-gb	kako
english-newzealand	kalaallisut
english-nz	kalenjin
english-unitedkingdom	kamba
english-unitedstates	kannada
english-us	kashmiri
english	kazakh
esperanto	khmer
estonian	kikuyu
ewe	kinyarwanda
ewondo	konkani
faroeese	korean
filipino	koyraborosenni
finnish	koyrachiini
french-be	kwasio
french-belgium	kyrgyz
french-ca	lakota
french-canada	langi
french-ch	lao
french-lu	latvian
french-luxembourg	lingala
french-switzerland	lithuanian
french	lowersorbian
friulian	lsorbian
fulah	lubakatanga

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



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  
norwegiannynorsk  
nswissgerman  
nuer  
nyankole  
nynorsk  
occitan  
oriya  
oromo  
ossetic  
pashto  
persian  
piedmontese  
polish  
polytonicgreek  
portuguese-br  
portuguese-brazil  
portuguese-portugal  
portuguese-pt  
portuguese  
punjabi-arab  
punjabi-arabic  
punjabi-gurmukhi  
punjabi-guru

punjabi  
quechua  
romanian  
romansh  
rombo  
rundi  
russian  
rwa  
sakha  
samburu  
samin  
sango  
sangu  
sanskrit-beng  
sanskrit-bengali  
sanskrit-deva  
sanskrit-devanagari  
sanskrit-gujarati  
sanskrit-gujr  
sanskrit-kannada  
sanskrit-knda  
sanskrit-malayalam  
sanskrit-mlym  
sanskrit-telu  
sanskrit-telugu  
sanskrit  
scottishgaelic  
sena  
serbian-cyrillic-bosniaherzegovina  
serbian-cyrillic-kosovo  
serbian-cyrillic-montenegro  
serbian-cyrillic  
serbian-cyrl-ba  
serbian-cyrl-me  
serbian-cyrl-xk  
serbian-cyrl  
serbian-latin-bosniaherzegovina  
serbian-latin-kosovo  
serbian-latin-montenegro  
serbian-latin  
serbian-latn-ba  
serbian-latn-me  
serbian-latn-xk  
serbian-latn  
serbian  
shambala  
shona  
sichuanyi  
sinhala  
slovak  
slovene  
slovenian  
soga  
somali  
spanish-mexico  
spanish-mx  
spanish  
standardmoroccantamazight

swahili	uyghur
swedish	uzbek-arab
swissgerman	uzbek-arabic
tachelhit-latin	uzbek-cyrillic
tachelhit-latn	uzbek-cyrl
tachelhit-tfng	uzbek-latin
tachelhit-tifinagh	uzbek-latn
tachelhit	uzbek
taita	vai-latin
tamil	vai-latn
tasawaq	vai-vai
telugu	vai-vaii
teso	vai
thai	vietnam
tibetan	vietnamese
tigrinya	vunjo
tongan	walser
turkish	welsh
turkmen	westernfrisian
ukenglish	yangben
ukrainian	yiddish
uppersorbian	yoruba
urdu	zarma
usenglish	zulu
usorbian	

---

### Modifying and adding values to ini files

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

## 1.14 Selecting fonts

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

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

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

The main purpose of `\babelfont` is to define at once in a multilingual document the fonts required by the different languages, with their corresponding language systems (script and language). So, if you load, say, 4 languages, `\babelfont{rm}{FreeSerif}` defines 4 fonts (with their variants, of course), which are switched with the language by babel. It is a tool to make things easier and transparent to the user.

Here *font-family* is `rm`, `sf` or `tt` (or newly defined ones, as explained below), and *font-name* is the same as in `fontspec` and the like.

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

On the other hand, if there is one or more languages in the optional argument, the font will be assigned to them, overriding the default one. Alternatively, you may set a font for a script – just precede its name (lowercase) with a star (eg, `*devanagari`). With this optional argument, the font is *not* yet defined, but just predeclared. This means you may define as

---

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

many fonts as you want ‘just in case’, because if the language is never selected, the corresponding `\babelfont` declaration is just ignored. Babel takes care of the font language and the font script when languages are selected (as well as the writing direction); see the recognized languages above. In most cases, you will not need *font-options*, which is the same as in fontspec, but you may add further key/value pairs if necessary.

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

LUATEX/XETEX

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{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 you 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`  $\{\langle\textit{language-name}\rangle\}\{\langle\textit{caption-name}\rangle\}\{\langle\textit{string}\rangle\}$

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

```
\setlocalecaption{english}{contents}{Table of Contents}
```

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

**NOTE** There are a few alternative methods:

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

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

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

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

```
\addto\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 add to `\extras⟨lang⟩`:

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

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

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

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

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

first loads `danish.ldf`, and then redefines the captions for danish (as provided by the `ini` file) and prevents hyphenation. The rest of the language definitions are not touched. Without the optional argument it just loads some additional tools if provided by the `ini` file, like extra counters.

## 1.16 Creating a language

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

`\babelprovide` [`⟨options⟩`] {`⟨language-name⟩`}

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

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

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

```
Package babel Warning: \chaptername not set for 'mylang'. Please,
(babel)                define it after the language has been loaded
(babel)                (typically in the preamble) with:
(babel)                \setlocalecaption{mylang}{chapter}{..}
(babel)                Reported on input line 26.
```

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

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

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

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

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

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

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

`\selectlanguage{arhinish}` or other selectors where necessary.

If the language has been loaded as an argument in `\documentclass` or `\usepackage`, then `\babelprovide` redefines the requested data.

**import=** *<language-tag>*

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

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

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

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

```
\babelprovide[import]{hungarian}
```

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

Besides `\today`, this option defines an additional command for dates: `\<language>date`, which takes three arguments, namely, year, month and day numbers. In fact, `\today` calls `\<language>today`, which in turn calls

`\<language>date{\the\year}{\the\month}{\the\day}`. **New 3.44** More convenient is usually `\localedate`, which prints the date for the current locale.

**captions=** *<language-tag>*

Loads only the strings. For example:

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

**hyphenrules=** *<language-list>*

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

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

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

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

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

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

**New 3.58** Another special value is `unhyphenated`, which 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.polutonic]{babel}
```

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

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

Remember there is an alternative syntax for the latter:

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

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

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

Same for \Alph.

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

**onchar=** ids | fonts | letters

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

**New 3.81** Option `letters` restricts the ‘actions’ to letters, in the 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: “یک, دو, and سه 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=**  $\langle$ base $\rangle$   $\langle$ shrink $\rangle$   $\langle$ stretch $\rangle$

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

**intrapenalty=**  $\langle$ penalty $\rangle$

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

**transforms=**  $\langle$ transform-list $\rangle$

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`  $\langle style \rangle \langle number \rangle$   
`\localecounter`  $\langle style \rangle \langle counter \rangle$

**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 avaiable styles in each language, see the list below):

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

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

The styles are:

**Ancient Greek** `lower.ancient, upper.ancient`  
**Amharic** `afar, agaw, ari, blin, dizi, gedeo, gumuz, hadiyya, harari, kaffa, kebona, kembata, konso, kunama, meen, oromo, saho, sidama, silti, tigre, wolaita, yemsa`  
**Arabic** `abjad, maghrebi.abjad`  
**Armenian** `lower.letter, upper.letter`  
**Belarusan, Bulgarian, Church Slavic, Macedonian, Serbian** `lower, upper`  
**Bangla** `alphabetic`  
**Central Kurdish** `alphabetic`  
**Chinese** `cjk-earthly-branch, cjk-heavenly-stem, circled.ideograph, parenthesized.ideograph, fullwidth.lower.alpha, fullwidth.upper.alpha`  
**Church Slavic (Glagolitic)** `letters`  
**Coptic** `epact, lower.letters`  
**French** `date.day` (mainly for internal use).  
**Georgian** `letters`  
**Greek** `lower.modern, upper.modern, lower.ancient, upper.ancient` (all with `keraia`)  
**Hebrew** `letters` (neither `geresh` nor `gershayim` yet)  
**Hindi** `alphabetic`  
**Italian** `lower.legal, upper.legal`  
**Japanese** `hiragana, hiragana.iroha, katakana, katakana.iroha, circled.katakana, informal, formal, cjk-earthly-branch, cjk-heavenly-stem, circled.ideograph, parenthesized.ideograph, fullwidth.lower.alpha, fullwidth.upper.alpha`  
**Khmer** `consonant`  
**Korean** `consonant, syllabe, hanja.informal, hanja.formal, hangul.formal, cjk-earthly-branch, cjk-heavenly-stem, circled.ideograph, parenthesized.ideograph, fullwidth.lower.alpha, fullwidth.upper.alpha`  
**Marathi** `alphabetic`  
**Persian** `abjad, alphabetic`  
**Russian** `lower, lower.full, upper, upper.full`  
**Syriac** `letters`  
**Tamil** `ancient`  
**Thai** `alphabetic`  
**Ukrainian** `lower, lower.full, upper, upper.full`

**New 3.45** In addition, native digits (in languages defining them) may be printed with the numeral style digits.

## 1.18 Dates

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

`\localedate` [`<calendar=.., variant=.., convert>`]{`<year>`}{`<month>`}{`<day>`}

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

calendar=hebrew and calendar=coptic). However, with the option convert it's converted (using internally the following command). Even with a certain calendar there may be variants. In Kurmanji the default variant prints something like *30. Çileyâ Pêşîn 2019*, but with variant=izafa it prints *31'ê Çileyâ 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	ethiopic	islamic-civil	persian
coptic	hebrew	islamic-umalqura	

The optional argument converts the given date, in the form '*<year>*-'*<month>*-'*<day>*'. Please, refer to the page on the news for 3.76 in the babel site for further details.

## 1.19 Accessing language info

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

**WARNING** Due to some internal inconsistencies in catcodes, it should *not* be used to test its value. Use iflang, by Heiko Oberdiek.

**\iflanguage** {*<language>*}{*<true>*}{*<false>*}

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

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

- `\babelhyphen{soft}` and `\babelhyphen{hard}` are self explanatory.
- `\babelhyphen{repeat}` inserts a hard hyphen which is repeated at the beginning of the next line, as done in languages like Polish, Portuguese and Spanish.
- `\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` `[\langle language \rangle, \langle language \rangle, ...]{\langle exceptions \rangle}`

**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} {<language>} ... \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 [ <language> , <language> , ... ] { <patterns> }`

**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:

<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.

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

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

Here are the transforms currently predefined. (A few may still require some fine-tuning. More to follow in future releases.)

Arabic	<code>transliteration.dad</code>	Applies the transliteration system devised by Yannis Haralambous for dad (simple and T <sub>E</sub> X-friendly). Not yet complete, but sufficient for most texts.
Croatian	<code>digraphs.ligatures</code>	Ligatures <i>DŽ, Dž, dž, LJ, Lj, lj, NJ, Nj, nj</i> . It assumes they exist. This is not the recommended way to make these transformations (the best way is with OTF features), but it can get you out of a hurry.
Czech, Polish, Portuguese, Slovak, Spanish	<code>hyphen.repeat</code>	Explicit hyphens behave like <code>\babelhyphen{repeat}</code> .
Czech, Polish, Slovak	<code>oneletter.nobreak</code>	Converts a space after a non-syllabic preposition or conjunction into a non-breaking space.
Finnish	<code>prehyphen.nobreak</code>	Line breaks just after hyphens prepended to words are prevented, like in “pakastekaapit ja -arkut”.
Greek	<code>diaeresis.hyphen</code>	Removes the diaeresis above iota and upsilon if hyphenated just before. It works with the three variants.
Greek	<code>transliteration.omega</code>	Although the provided combinations are not the full set, this transform follows the syntax of Omega: = 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	<code>sigma.final</code>	The transliteration system above does not convert the sigma at the end of a word (on purpose). This transform does it. To prevent the conversion (an abbreviation, for example), write "s.
Hindi, Sanskrit	<code>transliteration.hk</code>	The Harvard-Kyoto system to romanize Devanagari.
Hindi, Sanskrit	<code>punctuation.space</code>	Inserts a space before the following four characters: !?;.
Hungarian	<code>digraphs.hyphen</code>	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	<code>danda.nobreak</code>	Prevents a line break before a danda or double danda if there is a space. For Assamese, Bengali, Gujarati, Hindi, Kannada, Malayalam, Marathi, Odia, Tamil, Telugu.
Latin	<code>digraphs.ligatures</code>	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 ([*íú*]), the replacement could be {1|*íú*|*íú*}, which maps *í* to *í*, and *ú* to *ú*, so that the diaeresis is removed.

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

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

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

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

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

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

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

**New 3.44-3.52** It is similar to the latter, but (as its name implies) applied before hyphenation, which is particularly useful in transliterations. There are other differences: (1) the first argument is the locale instead of the name of the hyphenation patterns; (2) in the search patterns = has no special meaning, while | stands for an ordinary space; (3) in the replacement, discretionaries are not accepted.

See the description above for the optional argument.

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



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

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

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

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

**NOTE** With *luatex* there is another approach to make text transformations, with the function `fonts.handlers.otf.addfeature`, which adds new features to an OTF font (substitution and positioning). These features can be made language-dependent, and *babel* by default recognizes this setting if the font has been declared with `\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 `\getlocaleproperty`. It must be turned on explicitly for similar reasons to those explained above.

## 1.23 Selecting scripts

Currently `babel` provides no standard interface to select scripts, because they are best selected with either `\fontencoding` (low-level) or a language name (high-level). Even the Latin script may require different encodings (ie, sets of glyphs) depending on the language, and therefore such a switch would be in a sense incomplete.<sup>17</sup>

Some languages sharing the same script define macros to switch it (eg, `\textcyrillic`), but be aware they may also set the language to a certain default. Even the `babel` core defined `\textlatin`, but it was somewhat buggy because in some cases it messed up encodings and fonts (for example, if the main Latin encoding was `LY1`), and therefore it has been deprecated.<sup>18</sup>

```

\ensureascii {<text>}

```

**New 3.9i** This macro makes sure `<text>` is typeset with a LICR-savvy encoding in the ASCII range. It is used to redefine `\TeX` and `\LaTeX` so that they are correctly typeset even with LGR or X2 (the complete list is stored in `\BabelNonASCII`, which by default is LGR, X2, OT2, OT3, OT6, LHE, LWN, LMA, LMC, LMS, LMU, but you can modify it). So, in some sense it fixes the bug described in the previous paragraph.

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

<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.

example, if you load LY1 , LGR, then it is set to LY1, but if you load LY1 , T2A it is set to T2A. The symbol encodings TS1, T3, and TS3 are not taken into account, since they are not used for “ordinary” text (they are stored in \BabelNonText, used in some special cases when no Latin encoding is explicitly set).

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

## 1.24 Selecting directions

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

**WARNING** The current code for **text** in luatex should be considered essentially stable, but, of course, it is not bug-free and there can be improvements in the future, because setting bidi text has many subtleties (see for example <<https://www.w3.org/TR/html-bidi/>>). A basic stable version for other engines must wait. This applies to text; there is a basic support for **graphical** elements, including the picture environment (with pict2e) and pfg/tikz. Also, indexes and the like are under study, as well as math (there are progresses in the latter, including amsmath and mathtools too, but for example gathered may fail).

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

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

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

There are some package options controlling bidi writing.

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

**New 3.14** Selects the bidi algorithm to be used. With default the bidi mechanism is just activated (by default it is not), but every change must be marked up. In xetex and pdfTeX this is the only option.

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}
```

```

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}

```

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

```

\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

    Most Arabic speakers consider the two varieties to be two registers
    of one language, although the two registers can be referred to in
    Arabic as فصحى العصر \textit{fuṣḥā l-‘aṣr} (MSA) and
    فصحى التراث \textit{fuṣḥā t-turāth} (CA).

\end{document}

```

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

**NOTE** Boxes are “black boxes”. Numbers inside an `\hbox` (for example in a `\ref`) do not know anything about the surrounding chars. So, `\ref{A}-\ref{B}` are not rendered in the visual order A-B, but in the wrong one B-A (because the hyphen does not “see” the digits inside the `\hbox`’es). If you need `\ref` ranges, the best option is to define a dedicated macro like this (to avoid explicit direction changes in the body; here `\textthe` must be defined to select the main language):

```

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

```

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

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

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

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

**counters** required in all engines (except luatex with `bidi=basic`) to reorder section numbers and the like (eg, `\subsection`).`\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.

**New 3.84** Since `\thepage` is (indirectly) redefined, `makeindex` will reject many entries as invalid. With counters\* `babel` attempts to remove the conflicting macros.

**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 `r1` 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}\{(\{)\}
```

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

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

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

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

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

## 1.25 Language attributes

### `\languageattribute`

This is a user-level command, to be used in the preamble of a document (after `\usepackage[...]{babel}`), that declares which attributes are to be used for a given language. It takes two arguments: the first is the name of the language; the second, a (list of) attribute(s) to be used. Attributes must be set in the preamble and only once – they cannot be turned on and off. The command checks whether the language is known in this document and whether the attribute(s) are known for this language.

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

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

## 1.26 Hooks

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

**New 3.64** This is not the only way to inject code at those points. The events listed below can be used as a hook name in `\AddToHook` in the form `babel/⟨language-name⟩/⟨event-name⟩` (with \* it's applied to all languages), but there is a limitation, because the parameters passed with the babel mechanism are not allowed. The `\AddToHook` mechanism does *not* replace the current one in 'babel'. Its main advantage is you can reconfigure 'babel' even before loading it. See the example below.

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

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

**New 3.33** They may be also applied to a specific language with the optional argument; language-specific settings are executed after global ones.

Current events are the following; in some of them you can use one to three T<sub>E</sub>X parameters (#1, #2, #3), with the meaning given:

**adddialect** (language name, dialect name) Used by `luababel.def` to load the patterns if not preloaded.

**patterns** (language name, language with encoding) Executed just after the `\language` has been set. The second argument has the patterns name actually selected (in the form of either `lang:ENC` or `lang`).

**hyphenation** (language name, language with encoding) Executed locally just before exceptions given in `\babelhyphenation` are actually set.

**defaultcommands** Used (locally) in `\StartBabelCommands`.

**encodedcommands** (input, font encodings) Used (locally) in `\StartBabelCommands`. Both xetex and luatex make sure the encoded text is read correctly.

**stopcommands** Used to reset the above, if necessary.

**write** This event comes just after the switching commands are written to the aux file.

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

**afterextras** Just after executing `\extras⟨language⟩`. For example, the following deactivates shorthands in all languages:

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

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



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

**initiateactive** (char as active, char as other, original char) **New 3.9i** Executed just after a shorthand has been ‘initiated’. The three parameters are the same character with different catcodes: active, other (`\string’ed`) and the original one.

**afterreset** **New 3.9i** Executed when selecting a language just after `\originalTeX` is run and reset to its base value, before executing `\captions⟨language⟩` and `\date⟨language⟩`.

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

**everylanguage** (language) Executed before every language patterns are loaded.

**loadkernel** (file) By default just defines a few basic commands. It can be used to define different versions of them or to load a file.

**loadpatterns** (patterns file) Loads the patterns file. Used by `luababel.def`.

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

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

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

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

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

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

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

## 1.27 Languages supported by babel with ldf files

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

**Afrikaans** afrikaans

**Azerbaijani** azerbaijani

**Basque** basque

**Breton** breton

**Bulgarian** bulgarian

**Catalan** catalan

**Croatian** croatian

**Czech** czech

**Danish** danish

**Dutch** dutch

**English** english, USenglish, american, UKenglish, british, canadian, australian, newzealand

**Esperanto** esperanto



**Estonian** estonian  
**Finnish** finnish  
**French** french, francais, canadien, acadian  
**Galician** galician  
**German** austrian, german, germanb, ngerman, naustrian  
**Greek** greek, polutonikogreek  
**Hebrew** hebrew  
**Icelandic** icelandic  
**Indonesian** indonesian (bahasa, indon, bahasai)  
**Interlingua** interlingua  
**Irish Gaelic** irish  
**Italian** italian  
**Latin** latin  
**Lower Sorbian** lowersorbian  
**Malay** malay, melayu (bahasam)  
**North Sami** samin  
**Norwegian** norsk, nynorsk  
**Polish** polish  
**Portuguese** portuguese, brazilian (portuges, brazil)<sup>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** uppsorbian  
**Welsh** welsh

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

Most of them work out of the box, but some may require extra fonts, encoding files, a preprocessor or even a complete framework (like CJK or luatexja). For example, if you have got the velthuis/devnag package, you can create a file with extension .dn:

```

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

```

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

## 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).

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

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

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

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

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

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

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

## 1.29 Tweaking some features

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

**New 3.36** Sometimes you might need to disable some babel features. Currently this macro understands the following keys [to be documented], with values on or off:

<code>bidi.mirroring</code>	<code>linebreak.cjk</code>	<code>layout.lists</code>
<code>bidi.text</code>	<code>justify.arabic</code>	<code>autoload.bcp47</code>
<code>linebreak.sea</code>	<code>layout.tabular</code>	<code>bcp47.toname</code>

Other keys [to be documented] are:

<code>autoload.options</code>	<code>autoload.bcp47.options</code>	<code>select.write</code>
<code>autoload.bcp47.prefix</code>	<code>prehyphenation.disable</code>	<code>select.encoding</code>

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`),  $\LaTeX$  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 `hhline` 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 `hhline` (`babel`, now with the correct catcodes for `|` and `:`).

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

```
\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 apostrophes might not be taken into account. This is a limitation of T<sub>E</sub>X, not of babel. Alternatively, you may use `\usesorthands` to activate ' and `\definesorthand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).
- `\bibitem` is out of sync with `\selectlanguage` in the .aux file. The reason is `\bibitem` uses `\immediate` (and others, in fact), while `\selectlanguage` doesn't. There is a similar issue with floats, too. There is no known workaround.
- Babel does not take into account `\normalsfcodes` and (non-)French spacing is not always properly (un)set by languages. However, problems are unlikely to happen and therefore this part remains untouched in version 3.9 (but it is in the 'to do' list).
- Using a character mathematically active (ie, with math code "8000) as a shorthand can make T<sub>E</sub>X enter in an infinite loop in some rare cases. (Another issue in the 'to do' list, although there is a partial solution.)

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

**csquotes** Logical markup for quotes.

**iflang** Tests correctly the current language.

**hyphsubst** Selects a different set of patterns for a language.

**translator** An open platform for packages that need to be localized.

**siunitx** Typesetting of numbers and physical quantities.

**biblatex** Programmable bibliographies and citations.

**bicaption** Bilingual captions.

**babelbib** Multilingual bibliographies.

**microtype** Adjusts the typesetting according to some languages (kerning and spacing).

Ligatures can be disabled.

**substitutefont** Combines fonts in several encodings.

**mkpattern** Generates hyphenation patterns.

**tracklang** Tracks which languages have been requested.

**ucharclasses** (xetex) Switches fonts when you switch from one Unicode block to another.

**zhspacing** Spacing for CJK documents in xetex.

### 1.31 Current and future work

The current work is focused on the so-called complex scripts in luatex. In 8-bit engines, babel provided a basic support for bidi text as part of the style for Hebrew, but it is somewhat unsatisfactory and internally replaces some hardwired commands by other hardwired commands (generic changes would be much better).

Useful additions would be, for example, time, currency, addresses and personal names.<sup>21</sup>

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

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

Also interesting are differences in the sentence structure or related to it. For example, in Basque the number precedes the name (including chapters), in Hungarian "from (1)" is

<sup>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.

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

“(1)-ból”, but “from (3)” is “(3)-ból”, in Spanish an item labelled “3.<sup>o</sup>” may be referred to as either “ítem 3.<sup>o</sup>” or “3.<sup>er</sup> ítem”, and so on.

An option to manage bidirectional document layout in luatex (lists, footnotes, etc.) is almost finished, but xetex required more work. Unfortunately, proper support for xetex requires patching somehow lots of macros and packages (and some issues related to \specials remain, like color and hyperlinks), so babel resorts to the bidi package (by Vafa Khalighi). See the babel repository for a small example (xe-bidi).

### 1.32 Tentative and experimental code

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

#### Options for locales loaded on the fly

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

#### Labels

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

## 2 Loading languages with language.dat

T<sub>E</sub>X and most engines based on it (pdfT<sub>E</sub>X, xetex, ε-T<sub>E</sub>X, the main exception being luatex) require hyphenation patterns to be preloaded when a format is created (eg, L<sup>A</sup>T<sub>E</sub>X, XeL<sup>A</sup>T<sub>E</sub>X, pdfL<sup>A</sup>T<sub>E</sub>X). babel provides a tool which has become standard in many distributions and based on a “configuration file” named language.dat. The exact way this file is used depends on the distribution, so please, read the documentation for the latter (note also some distributions generate the file with some tool).

**New 3.9q** With luatex, however, patterns are loaded on the fly when requested by the language (except the “0th” language, typically english, which is preloaded always).<sup>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 T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X that the hyphenation patterns just processed have to be known under an alternative name. Here is an example:

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

<sup>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.

```
=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
```

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

A typical error when using babel is the following:

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

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

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

The *language definition files* (`ldf`) must conform to a number of conventions, because these files have to fill in the gaps left by the common code in `babel.def`, i. e., the definitions of the macros that produce texts. Also the language-switching possibility which has been built into the babel system has its implications.

The following assumptions are made:

- Some of the language-specific definitions might be used by plain  $\text{T}_{\text{E}}\text{X}$  users, so the files have to be coded so that they can be read by both  $\text{\LaTeX}$  and plain  $\text{T}_{\text{E}}\text{X}$ . The current format can be checked by looking at the value of the macro `\fmtname`.
- The common part of the babel system redefines a number of macros and environments (defined previously in the document style) to put in the names of macros that replace the previously hard-wired texts. These macros have to be defined in the language definition files.
- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are `\<lang>hyphenmins`, `\<lang>captions`, `\<lang>date`, `\<lang>extras` and `\<lang>noextras` (the last two may be left empty); where `<lang>` is either the name of the language definition file or the name of the  $\text{\LaTeX}$  option that is to be used. These macros and their functions are discussed below. You must define all or none for a language (or a dialect); defining, say, `\<lang>date` but not `\<lang>captions` does not raise an error but can lead to unexpected results.
- When a language definition file is loaded, it can define `\l@<lang>` to be a dialect of `\language0` when `\l@<lang>` is undefined.

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

- Language names must be all lowercase. If an unknown language is selected, babel will attempt setting it after lowercasing its name.
- The semantics of modifiers is not defined (on purpose). In most cases, they will just be simple separated options (eg, spanish), but a language might require, say, a set of options organized as a tree with suboptions (in such a case, the recommended separator is /).

Some recommendations:

- The preferred shorthand is ", which is not used in  $\LaTeX$  (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, `\babel@save` and `\babel@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.

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

- Babel ldf files may just interface a framework, as it happens often with Oriental languages/scripts. This framework is best placed in its own directory.

The following page provides a starting point for ldf files:

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

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

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

### 3.2 Basic macros

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

- \addlanguage** The macro `\addlanguage` is a non-outer version of the macro `\newlanguage`, defined in `plain.tex` version 3.x. Here “language” is used in the T<sub>E</sub>X sense of set of hyphenation patterns.
- \adddialect** The macro `\adddialect` can be used when two languages can (or must) use the same hyphenation patterns. This can also be useful for languages for which no patterns are preloaded in the format. In such cases the default behavior of the babel system is to define this language as a ‘dialect’ of the language for which the patterns were loaded as `\language0`. Here “language” is used in the T<sub>E</sub>X sense of set of hyphenation patterns.
- \<lang>hyphenmins** The macro `\<lang>hyphenmins` is used to store the values of the `\lefthyphenmin` and `\righthyphenmin`. Redefine this macro to set your own values, with two numbers corresponding to these two parameters. For example:

```
\renewcommand\spanishhyphenmins{34}
```

(Assigning `\lefthyphenmin` and `\righthyphenmin` directly in `\extras<lang>` has no effect.)

- \providehyphenmins** The macro `\providehyphenmins` should be used in the language definition files to set `\lefthyphenmin` and `\righthyphenmin`. This macro will check whether these parameters were provided by the hyphenation file before it takes any action. If these values have been already set, this command is ignored (currently, default pattern files do *not* set them).
- \captions<lang>** The macro `\captions<lang>` defines the macros that hold the texts to replace the original hard-wired texts.
- \date<lang>** The macro `\date<lang>` defines `\today`.
- \extras<lang>** The macro `\extras<lang>` contains all the extra definitions needed for a specific language. This macro, like the following, is a hook – you can add things to it, but it must not be used directly.
- \noextras<lang>** Because we want to let the user switch between languages, but we do not know what state T<sub>E</sub>X might be in after the execution of `\extras<lang>`, a macro that brings T<sub>E</sub>X into a predefined state is needed. It will be no surprise that the name of this macro is `\noextras<lang>`.
- \bbl@declare@ttribute** This is a command to be used in the language definition files for declaring a language attribute. It takes three arguments: the name of the language, the attribute to be defined, and the code to be executed when the attribute is to be used.
- \main@language** To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use `\main@language` instead of `\selectlanguage`. This will just store the name of the language, and the proper language will be activated at the start of the document.
- \ProvidesLanguage** The macro `\ProvidesLanguage` should be used to identify the language definition files. Its syntax is similar to the syntax of the L<sup>A</sup>T<sub>E</sub>X command `\ProvidesPackage`.
- \LdfInit** The macro `\LdfInit` performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the `@`-sign, preventing the `.ldf` file from being processed twice, etc.
- \ldf@quit** The macro `\ldf@quit` does work needed if a `.ldf` file was processed earlier. This includes



resetting the category code of the @-sign, preparing the language to be activated at `\begin{document}` time, and ending the input stream.

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

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

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

### 3.3 Skeleton

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

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

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

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

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

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

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

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

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

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

\EndBabelCommands

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

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



**NOTE** If for some reason you want to load a package in your style, you should be aware it cannot be done directly in the `ldf` file, but it can be delayed with `\AtEndOfPackage`. Macros from external packages can be used *inside* definitions in the `ldf` itself (for example, `\extras<language>`), but if executed directly, the code must be placed inside `\AtEndOfPackage`. A trivial example illustrating these points is:

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

### 3.4 Support for active characters

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

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

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

`\bbl@deactivate` `\bbl@activate` ‘switches on’ the active behavior of the character. `\bbl@deactivate` lets the active character expand to its former (mostly) non-active self.

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

`\bbl@add@special` The  $\TeX$ book states: “Plain  $\TeX$  includes a macro called `\dospecials` that is essentially a set macro, representing the set of all characters that have a special category code.” [4, p. 380]

`\bbl@remove@special` It is used to set text ‘verbatim’. To make this work if more characters get a special category code, you have to add this character to the macro `\dospecial`.  $\TeX$  adds another macro called `\@sanitize` representing the same character set, but without the curly braces. The macros `\bbl@add@special<char>` and `\bbl@remove@special<char>` add and remove the character `<char>` to these two sets.

### 3.5 Support for saving macro definitions

Language definition files may want to *redefine* macros that already exist. Therefore a mechanism for saving (and restoring) the original definition of those macros is provided. We provide two macros for this<sup>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`.

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

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

### 3.7 Macros common to a number of languages

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

### 3.8 Encoding-dependent strings

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

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

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

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

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

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

Encoding info is `charset=` followed by a `charset`, which if given sets how the strings should be translated to the internal representation used by the engine, typically `utf8`, which is the only value supported currently (default is no translations). Note `charset` is applied by

luatex and xetex when reading the file, not when the macro or string is used in the document.

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

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

The *category* is either captions, date or extras. You must stick to these three categories, even if no error is raised when using other name.<sup>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\monthviiname{August}
\SetString\monthixname{September}
\SetString\monthxname{Oktober}
\SetString\monthxiname{November}
\SetString\monthxiiname{Dezenber}
\SetString\today{\number\day.~%
\csname month\romannumeral\month name\endcsname\space
```

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

```

\number\year}

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

\EndBabelCommands

```

When used in ldf files, previous values of `\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 `\date \langle language \rangle` exists).

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

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  $\TeX$ , we can set for Turkish:

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

```

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

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

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

\EndBabelCommands

```

(Note the mapping for OT1 is not complete.)

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

**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<language>`.

## Part II

# Source code

babel is being developed incrementally, which means parts of the code are under development and therefore incomplete. Only documented features are considered complete. In other words, use babel only as documented (except, of course, if you want to explore and test them – you can post suggestions about multilingual issues to [kadingira@tug.org](mailto:kadingira@tug.org) on <http://tug.org/mailman/listinfo/kadingira>).

## 4 Identification and loading of required files

*Code documentation is still under revision.*

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

The babel package after unpacking consists of the following files:

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

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

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

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

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

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

## 5 locale directory

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

Most of them are essentially finished (except bugs and mistakes, of course). Some of them are still incomplete (but they will be usable), and there are some omissions (eg, Latin and polytonic Greek, and there are no geographic areas in Spanish). Hindi, French, Occitan and Breton will show a warning related to dates. Not all include LICR variants.

This is a preliminary documentation.

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

**charset** the encoding used in the ini file.

**version** of the ini file

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

**encodings** a descriptive list of font encodings.

**[captions]** section of captions in the file charset

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

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

Keys may be further qualified in a particular language with a suffix starting with an uppercase letter. It can be just a letter (eg, `babel.name.A`, `babel.name.B`) or a name (eg, `date.long.Nominative`, `date.long.Formal`, but no language is currently using the latter). *Multi-letter* qualifiers are forward compatible in the sense they won’t conflict with new “global” keys (which start always with a

lowercase case). There is an exception, however: the section counters has been devised to have arbitrary keys, so you can add lowercased keys if you want.

## 6 Tools

```
1 <<version=3.84.2989>>
2 <<date=2023/01/21>>
```

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

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

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

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

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

`\bbl@afterelse` `\bbl@afterfi` Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the `\else` and `\fi` parts of an `\if`-statement<sup>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{%
```

<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.

```

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

```

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

```

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

```

`\bblexp@ifunset` To check if a macro is defined, we create a new macro, which does the same as `\ifundefined`. However, in an  $\epsilon$ -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\bblexp@ifunset#1{%
58 \expandafter\ifx\csname#1\endcsname\relax
59 \expandafter\@firstoftwo
60 \else
61 \expandafter\@secondoftwo
62 \fi}
63 \bblexp@ifunset{ifcsname}%
64 {}%
65 {\gdef\bblexp@ifunset#1{%
66 \ifcsname#1\endcsname
67 \expandafter\ifx\csname#1\endcsname\relax
68 \bblexp@afterelse\expandafter\@firstoftwo
69 \else
70 \bblexp@afterfi\expandafter\@secondoftwo
71 \fi
72 \else
73 \expandafter\@firstoftwo
74 \fi}}
75 \endgroup

```

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

```

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

```

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



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

```

81 \def\bb1@forkv#1#2{%
82   \def\bb1@kvcmd##1##2##3{#2}%
83   \bb1@kvnext#1,\@nil,}
84 \def\bb1@kvnext#1,{%
85   \ifx\@nil#1\relax\else
86     \bb1@ifblank{#1}{\bb1@forkv@eq#1=\@empty=\@nil{#1}}%
87     \expandafter\bb1@kvnext
88   \fi}
89 \def\bb1@forkv@eq#1=#2=#3\@nil#4{%
90   \bb1@trim\def\bb1@forkv@a{#1}%
91   \bb1@trim{\expandafter\bb1@kvcmd\expandafter{\bb1@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\bb1@vforeach#1#2{%
93   \def\bb1@forcmd##1{#2}%
94   \bb1@fornext#1,\@nil,}
95 \def\bb1@fornext#1,{%
96   \ifx\@nil#1\relax\else
97     \bb1@ifblank{#1}{\bb1@trim\bb1@forcmd{#1}}%
98     \expandafter\bb1@fornext
99   \fi}
100 \def\bb1@foreach#1{\expandafter\bb1@vforeach\expandafter{#1}}

```

**\bb1@replace** Returns implicitly \toks@ with the modified string.

```

101 \def\bb1@replace#1#2#3{% in #1 -> repl #2 by #3
102   \toks@{}}%
103   \def\bb1@replace@aux##1#2##2#2{%
104     \ifx\bb1@nil##2%
105       \toks@\expandafter{\the\toks@##1}%
106     \else
107       \toks@\expandafter{\the\toks@##1#3}%
108       \bb1@afterfi
109       \bb1@replace@aux##2#2%
110     \fi}%
111   \expandafter\bb1@replace@aux#1#2\bb1@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 \bb1@TG@@date, and also fails if there are macros with spaces, because they are retokenized). It may change! (or even merged with \bb1@replace; I'm not sure ckecking the replacement is really necessary or just paranoia).

```

113 \ifx\detokenize\undefined\else % Unused macros if old Plain TeX
114   \bb1@exp{\def\\bb1@parsedef##1\detokenize{macro:}}#2->#3\relax{%
115     \def\bb1@tempa{#1}%
116     \def\bb1@tempb{#2}%
117     \def\bb1@tempe{#3}}
118   \def\bb1@sreplace#1#2#3{%
119     \begingroup
120     \expandafter\bb1@parsedef\meaning#1\relax
121     \def\bb1@tempc{#2}%
122     \edef\bb1@tempc{\expandafter\strip@prefix\meaning\bb1@tempc}%
123     \def\bb1@tempd{#3}%
124     \edef\bb1@tempd{\expandafter\strip@prefix\meaning\bb1@tempd}%
125     \bb1@xin@{\bb1@tempc}{\bb1@tempe}% If not in macro, do nothing
126     \ifin@
127       \bb1@exp{\bb1@replace\\bb1@tempe{\bb1@tempc}{\bb1@tempd}}%
128       \def\bb1@tempc{% Expanded an executed below as 'uplevel'
129         \\makeatletter % "internal" macros with @ are assumed
130         \\scantokens{%

```

```

131          \bbl@tempa\\@namedef{\bbl@stripslash#1}\bbl@tempb{\bbl@tempe}}%
132          \catcode64=\the\catcode64\relax}% Restore @
133      \else
134          \let\bbl@tempc\@empty % Not \relax
135      \fi
136      \bbl@exp{%          For the 'uplevel' assignments
137  \endgroup
138      \bbl@tempc}} % empty or expand to set #1 with changes
139 \fi

```

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

```

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

```

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

```

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

```

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

```

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

```

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`).

```

181 \def\bb1@extras@wrap#1#2#3{% 1:in-test, 2:before, 3:after
182   \toks@\expandafter\expandafter\expandafter{%
183     \csname extras\language\endcsname}%
184   \bb1@exp{\in@{#1}\the\toks@}}%
185   \ifin@ \else
186     \@temptokena{#2}%
187     \edef\bb1@tempc{\the\@temptokena\the\toks@}%
188     \toks@\expandafter{\bb1@tempc#3}%
189     \expandafter\edef\csname extras\language\endcsname{\the\toks@}%
190   \fi}
191 \<</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  $\TeX$ .

```

192 \<<*Make sure ProvidesFile is defined>> ≡
193 \ifx\ProvidesFile\@undefined
194   \def\ProvidesFile#1[#2 #3 #4]{%
195     \wlog{File: #1 #4 #3 <#2>}%
196     \let\ProvidesFile\@undefined}
197 \fi
198 \<</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.

```

199 \<<*Define core switching macros>> ≡
200 \ifx\language\@undefined
201   \csname newcount\endcsname\language
202 \fi
203 \<</Define core switching macros>>

```

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

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

```

204 \<<*Define core switching macros>> ≡
205 \countdef\last@language=19
206 \def\addlanguage{\csname newlanguage\endcsname}
207 \<</Define core switching macros>>

```

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

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

```

208 \<*package>
209 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
210 \ProvidesPackage{babel}[\<<date>> \<<version>>] The Babel package]

```

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

```

211 \@ifpackagewith{babel}{debug}
212   {\providecommand\bb1@trace[1]{\message{^^J[ #1 ]}}%
213   \let\bb1@debug\@firstofone
214   \ifx\directlua\@undefined\else

```

```

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

```

245 <Basic macros>
246 \@ifpackagewith{babel}{silent}
247   {\let\bbl@info@gobble
248    \let\bbl@infowarn@gobble
249    \let\bbl@warning@gobble}
250   {}
251 %
252 \def\AfterBabelLanguage#1{%
253   \global\expandafter\bbl@add\csname#1.ldf-h@k\endcsname}%

```

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

```

254 \ifx\bbl@languages@undefined\else
255   \begingroup
256     \catcode`\^^I=12
257     \@ifpackagewith{babel}{showlanguages}{%
258       \begingroup
259         \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
260         \wlog{<*languages>}%
261         \bbl@languages
262         \wlog{</languages>}%
263       \endgroup}{}
264     \endgroup
265   \def\bbl@elt#1#2#3#4{%
266     \ifnum#2=\z@
267       \gdef\bbl@nulllanguage{#1}%
268     \def\bbl@elt##1##2##3##4{%

```

```

269 \fi}%
270 \bbl@languages
271 \fi%

```

### 6.3 base

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

```

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

```

292 \bbl@trace{key=value and another general options}
293 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
294 \def\bbl@tempb#1.#2{% Remove trailing dot
295 #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
296 \def\bbl@tempd#1.#2\@nnil{% TODO. Refactor lists?
297 \ifx\@empty#2%
298 \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
299 \else
300 \in@{,provide=}{, #1}%
301 \ifin@
302 \edef\bbl@tempc{%
303 \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
304 \else
305 \in@{=}{#1}%
306 \ifin@
307 \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
308 \else
309 \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
310 \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
311 \fi
312 \fi
313 \fi}
314 \let\bbl@tempc\@empty
315 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
316 \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.

```

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

```

337 \let\bbl@opt@shorthands\@nnil
338 \let\bbl@opt@config\@nnil
339 \let\bbl@opt@main\@nnil
340 \let\bbl@opt@headfoot\@nnil
341 \let\bbl@opt@layout\@nnil
342 \let\bbl@opt@provide\@nnil

```

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

```

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

```

354 \let\bbl@language@opts\@empty
355 \DeclareOption*{%
356   \bbl@xin@{\string=}{\CurrentOption}%
357   \ifin@
358     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
359   \else
360     \bbl@add@list\bbl@language@opts{\CurrentOption}%
361   \fi}

```

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

```

362 \ProcessOptions*

```

```

363 \ifx\bbbl@opt@provide\@nnil
364 \let\bbbl@opt@provide\@empty %%% MOVE above
365 \else
366 \chardef\bbbl@iniflag\@ne
367 \bbbl@exp{\bbbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
368 \in{,provide,}{, #1,}%
369 \ifin@
370 \def\bbbl@opt@provide{#2}%
371 \bbbl@replace\bbbl@opt@provide{;}{,}%
372 \fi}
373 \fi
374 %

```

## 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 `\bbbl@ifshorthand` is always true, and it is always false if `shorthands` is empty. Also, some code makes sense only with `shorthands=...`

```

375 \bbbl@trace{Conditional loading of shorthands}
376 \def\bbbl@sh@string#1{%
377 \ifx#1\@empty\else
378 \ifx#1t\string~%
379 \else\ifx#1c\string,%
380 \else\string#1%
381 \fi\fi
382 \expandafter\bbbl@sh@string
383 \fi}
384 \ifx\bbbl@opt@shorthands\@nnil
385 \def\bbbl@ifshorthand#1#2#3{#2}%
386 \else\ifx\bbbl@opt@shorthands\@empty
387 \def\bbbl@ifshorthand#1#2#3{#3}%
388 \else

```

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

```

389 \def\bbbl@ifshorthand#1{%
390 \bbbl@xin@{\string#1}{\bbbl@opt@shorthands}%
391 \ifin@
392 \expandafter\@firstoftwo
393 \else
394 \expandafter\@secondoftwo
395 \fi}

```

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

```

396 \edef\bbbl@opt@shorthands{%
397 \expandafter\bbbl@sh@string\bbbl@opt@shorthands\@empty}%

```

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

```

398 \bbbl@ifshorthand{'}%
399 {\PassOptionsToPackage{activeacute}{babel}}{}
400 \bbbl@ifshorthand{`}%
401 {\PassOptionsToPackage{activegrave}{babel}}{}
402 \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.

```

403 \ifx\bbbl@opt@headfoot\@nnil\else
404 \g@addto@macro\@resetactivechars{%
405 \set@typeset@protect
406 \expandafter\select@language@x\expandafter{\bbbl@opt@headfoot}%
407 \let\protect\noexpand}
408 \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.

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

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

```
413 \bbl@trace{Defining IfBabelLayout}
414 \ifx\bbl@opt@layout\@nnil
415   \newcommand\IfBabelLayout[3]{#3}%
416 \else
417   \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
418     \in@{,layout,}{, #1,}%
419     \ifin@
420       \def\bbl@opt@layout{#2}%
421       \bbl@replace\bbl@opt@layout{ }{.}%
422     \fi}
423   \newcommand\IfBabelLayout[1]{%
424     \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
425     \ifin@
426       \expandafter\@firstoftwo
427     \else
428       \expandafter\@secondoftwo
429     \fi}
430 \fi
431 \</package>
432 \<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*.

```
433 \ifx\ldf@quit\@undefined\else
434 \endinput\fi % Same line!
435 \<Make sure ProvidesFile is defined>
436 \ProvidesFile{babel.def}[\<date>] [\<version>] Babel common definitions]
437 \ifx\AtBeginDocument\@undefined % TODO. change test.
438   \<Emulate LaTeX>
439 \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.

```
440 \</core>
441 \<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.

```
442 \def\bbl@version{\<version>}
443 \def\bbl@date{\<date>}
444 \<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.

```
445 \def\adddialect#1#2{%
446   \global\chardef#1#2\relax
```



```

447 \bbl@usehooks{adddialect}{#1}{#2}}%
448 \begingroup
449 \count#1\relax
450 \def\bbl@elt##1##2##3##4{%
451   \ifnum\count@=##2\relax
452     \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
453     \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
454       set to \expandafter\string\csname l@##1\endcsname\%
455       (\string\language\the\count@). Reported}%
456     \def\bbl@elt####1####2####3####4{%
457       \fi}%
458     \bbl@cs{languages}%
459   \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.

```

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

```

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

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

```

478 \def\bbl@bcpcase#1#2#3#4\@#5{%
479   \ifx\@empty#3%
480     \uppercase{\def#5{#1#2}}%
481   \else
482     \uppercase{\def#5{#1}}%
483     \lowercase{\edef#5{#5#2#3#4}}%
484   \fi}
485 \def\bbl@bcpllookup#1-#2-#3-#4\@#5{%
486   \let\bbl@bcp\relax
487   \lowercase{\def\bbl@tempa{#1}}%
488   \ifx\@empty#2%
489     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
490   \else\ifx\@empty#3%
491     \bbl@bcpcase#2\@empty\@empty\@#5\bbl@tempb
492     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
493     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
494     {}%
495   \ifx\bbl@bcp\relax
496     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%

```

```

497 \fi
498 \else
499 \bbl@bcp@case#2\@empty\@empty\@@\bbl@tempb
500 \bbl@bcp@case#3\@empty\@empty\@@\bbl@tempc
501 \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
502 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}}%
503 {}%
504 \ifx\bbl@bcp\relax
505 \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
506 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}}%
507 {}%
508 \fi
509 \ifx\bbl@bcp\relax
510 \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
511 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}}%
512 {}%
513 \fi
514 \ifx\bbl@bcp\relax
515 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}}}%
516 \fi
517 \fi\fi}
518 \let\bbl@initoload\relax
519 \def\bbl@provide@locale{%
520 \ifx\babelprovide\@undefined
521 \bbl@error{For a language to be defined on the fly 'base'\%
522 is not enough, and the whole package must be\%
523 loaded. Either delete the 'base' option or\%
524 request the languages explicitly}%
525 {See the manual for further details.}%
526 \fi
527 \let\bbl@auxname\language\language % Still necessary. TODO
528 \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
529 {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
530 \ifbbl@bcp@allowed
531 \expandafter\ifx\csname date\language\endcsname\relax
532 \expandafter
533 \bbl@bcp@lookup\language-\@empty-\@empty-\@empty\@@
534 \ifx\bbl@bcp\relax\else % Returned by \bbl@bcp@lookup
535 \edef\language{\bbl@bcp@prefix\bbl@bcp}%
536 \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
537 \expandafter\ifx\csname date\language\endcsname\relax
538 \let\bbl@initoload\bbl@bcp
539 \bbl@exp{\bbl@babelprovide[\bbl@autoload@bcptoptions]{\language}}}%
540 \let\bbl@initoload\relax
541 \fi
542 \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
543 \fi
544 \fi
545 \fi
546 \expandafter\ifx\csname date\language\endcsname\relax
547 \IfFileExists{babel-\language.tex}%
548 {\bbl@exp{\bbl@babelprovide[\bbl@autoload@options]{\language}}}%
549 {}%
550 \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.

```
551 \def\iflanguage#1{%
552   \bbl@iflanguage{#1}{%
553     \ifnum\csname l@#1\endcsname=\language
554     \expandafter\@firstoftwo
```

```

555 \else
556 \expandafter\@secondoftwo
557 \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.

```

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

```

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

```

562 \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.

```

563 \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.

```

564 \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`

```

565 \def\bbl@push@language{%
566 \ifx\language\@undefined\else
567 \ifx\currentgrouplevel\@undefined
568 \xdef\bbl@language@stack{\language+\bbl@language@stack}%
569 \else
570 \ifnum\currentgrouplevel=\z@
571 \xdef\bbl@language@stack{\language+}%
572 \else
573 \xdef\bbl@language@stack{\language+\bbl@language@stack}%
574 \fi
575 \fi
576 \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`.

```

577 \def\bbl@pop@lang#1+#2@@{%
578 \edef\language{#1}%
579 \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).

```
580 \let\bbl@ifrestoring\@secondoftwo
581 \def\bbl@pop@language{%
582   \expandafter\bbl@pop@lang\bbl@language@stack\@@
583   \let\bbl@ifrestoring\@firstoftwo
584   \expandafter\bbl@set@language\expandafter{\language}%
585   \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).

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

The unprotected part of `\selectlanguage`.

```
604 \expandafter\def\csname selectlanguage \endcsname#1{%
605   \ifnum\bbl@hymapsel=\ccclv\let\bbl@hymapsel\tw@\fi
606   \bbl@push@language
607   \aftergroup\bbl@pop@language
608   \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 `laTeX`, is to avoid the `\write` altogether when not needed).

```
609 \def\BabelContentsFiles{toc,lof,lot}
610 \def\bbl@set@language#1{% from selectlanguage, pop@
611   % The old buggy way. Preserved for compatibility.
612   \edef\language{%
613     \ifnum\escapechar=\expandafter\string#1\@empty
614     \else\string#1\@empty\fi}%
615 }
```

```

615 \ifcat\relax\noexpand#1%
616 \expandafter\ifx\csname date\language\endcsname\relax
617 \edef\language{#1}%
618 \let\locale\language
619 \else
620 \bbl@info{Using '\string\language' instead of 'language' is\\%
621 deprecated. If what you want is to use a\\%
622 macro containing the actual locale, make\\%
623 sure it does not not match any language.\\%
624 Reported}%
625 \ifx\scantokens\undefined
626 \def\locale{??}%
627 \else
628 \scantokens\expandafter{\expandafter
629 \def\expandafter\locale\expandafter{\language}}%
630 \fi
631 \fi
632 \else
633 \def\locale{#1}% This one has the correct catcodes
634 \fi
635 \select@language{\language}%
636 % write to aux
637 \expandafter\ifx\csname date\language\endcsname\relax\else
638 \if@filesw
639 \ifx\babel@aux\@gobbles\else % Set if single in the first, redundant
640 \bbl@savelastskip
641 \protected@write\auxout{}\string\babel@aux{\bbl@auxname{}}%
642 \bbl@restorelastskip
643 \fi
644 \bbl@usehooks{write}{}%
645 \fi
646 \fi}
647 %
648 \let\bbl@restorelastskip\relax
649 \let\bbl@savelastskip\relax
650 %
651 \newif\ifbbl@bcpallowed
652 \bbl@bcpallowedfalse
653 \def\select@language#1{% from set@, babel@aux
654 \ifx\bbl@select@name\empty
655 \def\bbl@select@name{select}%
656 % set hmap
657 \fi
658 \ifnum\bbl@hmapset=\@cclv\chardef\bbl@hmapset4\relax\fi
659 % set name
660 \edef\language{#1}%
661 \bbl@fixname\language
662 % TODO. name@map must be here?
663 \bbl@provide@locale
664 \bbl@iflanguage\language{%
665 \let\bbl@select@type\z@
666 \expandafter\bbl@switch\expandafter{\language}}%
667 \def\babel@aux#1#2{%
668 \select@language{#1}%
669 \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
670 \writefile{##1}{\babel@toc{#1}{#2}\relax}}% TODO - plain?
671 \def\babel@toc#1#2{%
672 \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.

```

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

```

```

728 \let\bbl@hymapsel\@cclv
729 % hyphenation - select rules
730 \ifnum\csname l@language\endcsname=\l@unhyphenated
731 \edef\bbl@tempa{u}%
732 \else
733 \edef\bbl@tempa{\bbl@cl{\lnbrk}}%
734 \fi
735 % linebreaking - handle u, e, k (v in the future)
736 \bbl@xin@{/u}{/\bbl@tempa}%
737 \ifin@ \else\bbl@xin@{/e}{/\bbl@tempa}\fi % elongated forms
738 \ifin@ \else\bbl@xin@{/k}{/\bbl@tempa}\fi % only kashida
739 \ifin@ \else\bbl@xin@{/p}{/\bbl@tempa}\fi % padding (eg, Tibetan)
740 \ifin@ \else\bbl@xin@{/v}{/\bbl@tempa}\fi % variable font
741 \ifin@
742 % unhyphenated/kashida/elongated/padding = allow stretching
743 \language\l@unhyphenated
744 \babel@savevariable\emergencystretch
745 \emergencystretch\maxdimen
746 \babel@savevariable\hbadness
747 \hbadness\@M
748 \else
749 % other = select patterns
750 \bbl@patterns{#1}%
751 \fi
752 % hyphenation - mins
753 \babel@savevariable\lefthyphenmin
754 \babel@savevariable\righthyphenmin
755 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
756 \set@hyphenmins\tw@\thr@@\relax
757 \else
758 \expandafter\expandafter\expandafter\set@hyphenmins
759 \csname #1hyphenmins\endcsname\relax
760 \fi
761 \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.

```

762 \long\def\otherlanguage#1{%
763 \def\bbl@selectorname{other}%
764 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
765 \csname selectlanguage \endcsname{#1}%
766 \ignorespaces}

```

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

```

767 \long\def\endotherlanguage{%
768 \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`.

```

769 \expandafter\def\csname otherlanguage*\endcsname{%
770 \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
771 \def\bbl@otherlanguage@s[#1]#2{%
772 \def\bbl@selectorname{other*}%
773 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
774 \def\bbl@select@opts{#1}%
775 \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”.

```
776 \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`.

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

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

```
805 \def\foreign@language#1{%
806   % set name
807   \edef\language#1}%
808   \ifbbl@usedatagroup
809     \bbl@add\bbl@select@opts{,date,}%
```



```

810 \bbl@usedategroupfalse
811 \fi
812 \bbl@fixname\language
813 % TODO. name@map here?
814 \bbl@provide@locale
815 \bbl@iflanguage\language{%
816 \let\bbl@select@type\@ne
817 \expandafter\bbl@switch\expandafter{\language}}

```

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

```

818 \def\IfBabelSelectorTF#1{%
819 \bbl@xin@{\bbl@selectorname,}{,\zap@space#1 \@empty,}%
820 \ifin@
821 \expandafter\@firstoftwo
822 \else
823 \expandafter\@secondoftwo
824 \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.

```

825 \let\bbl@hyphlist\@empty
826 \let\bbl@hyphenation@\relax
827 \let\bbl@pttnlist\@empty
828 \let\bbl@patterns@\relax
829 \let\bbl@hymapsel=\ccclv
830 \def\bbl@patterns#1{%
831 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
832 \csname l@#1\endcsname
833 \edef\bbl@tempa{#1}%
834 \else
835 \csname l@#1:\f@encoding\endcsname
836 \edef\bbl@tempa{#1:\f@encoding}%
837 \fi
838 \@expandtwoargs\bbl@usehooks{patterns}{#{1}}{\bbl@tempa}}%
839 % > luatex
840 \@ifundefined{bbl@hyphenation@}{% Can be \relax!
841 \begingroup
842 \bbl@xin@{\number\language,}{,\bbl@hyphlist}%
843 \ifin@
844 \else
845 \expandafter\bbl@usehooks{hyphenation}{#{1}}{\bbl@tempa}}%
846 \hyphenation{%
847 \bbl@hyphenation@
848 \@ifundefined{bbl@hyphenation@#1}%
849 \@empty
850 {\space\csname bbl@hyphenation@#1\endcsname}}%
851 \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
852 \fi
853 \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*`.

```

853 \def\hyphenrules#1{%
854 \edef\bbl@tempf{#1}%
855 \bbl@fixname\bbl@tempf
856 \bbl@iflanguage\bbl@tempf{%

```

```

857 \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
858 \ifx\languageshorthands\@undefined\else
859 \languageshorthands{none}%
860 \fi
861 \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
862 \set@hyphenmins\tw@\thr@\relax
863 \else
864 \expandafter\expandafter\expandafter\set@hyphenmins
865 \csname\bbl@tempf hyphenmins\endcsname\relax
866 \fi}}
867 \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.

```

868 \def\providehyphenmins#1#2{%
869 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
870 \@namedef{#1hyphenmins}{#2}%
871 \fi}

```

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

```

872 \def\set@hyphenmins#1#2{%
873 \lefthyphenmin#1\relax
874 \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.

```

875 \ifx\ProvidesFile\@undefined
876 \def\ProvidesLanguage#1[#2 #3 #4]{%
877 \wlog{Language: #1 #4 #3 <#2>}%
878 }
879 \else
880 \def\ProvidesLanguage#1{%
881 \begingroup
882 \catcode`\ 10 %
883 \@makeother\/%
884 \@ifnextchar[%]
885 {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
886 \def\@provideslanguage#1[#2]{%
887 \wlog{Language: #1 #2}%
888 \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
889 \endgroup}
890 \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`.

```

891 \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.

```

892 \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’:

```

893 \providecommand\setlocale{%
894 \bbl@error
895 {Not yet available}%
896 {Find an armchair, sit down and wait}}
897 \let\uselocale\setlocale
898 \let\locale\setlocale

```

```

899 \let\selectlocale\setlocale
900 \let\textlocale\setlocale
901 \let\textlanguage\setlocale
902 \let\language\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.

```

903 \edef\bbl@nulllanguage{\string\language=0}
904 \def\bbl@nocaption{\protect\bbl@nocaption@i}
905 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
906   \global\@namedef{#2}{\textbf{?#1?}}%
907   \@nameuse{#2}%
908   \edef\bbl@tempa{#1}%
909   \bbl@sreplace\bbl@tempa{name}}}%
910 \bbl@warning{%
911   \@backslashchar#1 not set for '\language'. Please,\\%
912   define it after the language has been loaded\\%
913   (typically in the preamble) with:\\%
914   \string\setlocalecaption{\language}{\bbl@tempa}{..}\\%
915   Feel free to contribute on github.com/latex3/babel.\\%
916   Reported}}
917 \def\bbl@tentative{\protect\bbl@tentative@i}
918 \def\bbl@tentative@i#1{%
919   \bbl@warning{%
920     Some functions for '#1' are tentative.\\%
921     They might not work as expected and their behavior\\%
922     could change in the future.\\%
923     Reported}}
924 \def\@nolanerr#1{%
925   \bbl@error
926   {You haven't defined the language '#1' yet.\\%
927     Perhaps you misspelled it or your installation\\%
928     is not complete}%
929   {Your command will be ignored, type <return> to proceed}}
930 \def\@nopatterns#1{%
931   \bbl@warning
932   {No hyphenation patterns were preloaded for\\%
933     the language '#1' into the format.\\%
934     Please, configure your TeX system to add them and\\%
935     rebuild the format. Now I will use the patterns\\%
936     preloaded for \bbl@nulllanguage\space instead}}
937 \let\bbl@usehooks\@gobbletwo
938 \ifx\bbl@onlyswitch\@empty\endinput\fi
939 % Here ended switch.def

```

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

```

940 \ifx\directlua\@undefined\else
941   \ifx\bbl@luapatterns\@undefined
942     \input luababel.def
943   \fi
944 \fi
945 <Basic macros>

```

```

946 \bbl@trace{Compatibility with language.def}
947 \ifx\bbl@languages\@undefined
948   \ifx\directlua\@undefined
949     \openin1 = language.def % TODO. Remove hardcoded number
950     \ifeof1
951       \closein1
952       \message{I couldn't find the file language.def}
953     \else
954       \closein1
955       \begingroup
956         \def\addlanguage#1#2#3#4#5{%
957           \expandafter\ifx\csname lang@#1\endcsname\relax\else
958             \global\expandafter\let\csname l@#1\expandafter\endcsname
959               \csname lang@#1\endcsname
960           \fi}%
961         \def\uselanguage#1{%
962           \input language.def
963         \endgroup
964       \fi
965     \fi
966     \chardef\l@english\z@
967 \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.

```

968 \def\addto#1#2{%
969   \ifx#1\@undefined
970     \def#1{#2}%
971   \else
972     \ifx#1\relax
973       \def#1{#2}%
974     \else
975       {\toks@\expandafter{#1#2}%
976        \xdef#1{\the\toks@}}%
977     \fi
978   \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.

```

979 \def\bbl@withactive#1#2{%
980   \begingroup
981   \lccode`~=#2\relax
982   \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`.

```

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

```

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

```

988 \def\bbl@redefine@long#1{%
989   \edef\bbl@tempa{\bbl@stripslash#1}%
990   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
991   \long\expandafter\def\csname\bbl@tempa\endcsname{
992     \@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`.

```

993 \def\bbl@redefineroobust#1{%
994   \edef\bbl@tempa{\bbl@stripslash#1}%
995   \bbl@ifunset{\bbl@tempa\space}%
996   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
997    \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
998   {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
999   \namedef{\bbl@tempa\space}}
1000 \@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.

```

1001 \bbl@trace{Hooks}
1002 \newcommand\AddBabelHook[3][{}]{%
1003   \bbl@ifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1004   \def\bbl@tempa##1,##2,##3\@empty{\def\bbl@tempb{##2}}%
1005   \expandafter\bbl@tempa\bbl@evargs,##3,\@empty
1006   \bbl@ifunset{\bbl@ev@#2@#3@#1}%
1007     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1008     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1009   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1010 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1011 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1012 \def\bbl@usehooks#1#2{%
1013   \ifx\UseHook\undefined\else\UseHook{babel/*/#1}\fi
1014   \def\bbl@elth##1{%
1015     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@#2}}%
1016     \bbl@cs{ev@#1@}%
1017     \ifx\language\undefined\else % Test required for Plain (?)
1018       \ifx\UseHook\undefined\else\UseHook{babel/\language/#1}\fi
1019       \def\bbl@elth##1{%
1020         \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1@#2}}%
1021         \bbl@cl{ev@#1@}%
1022       \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).

```

1023 \def\bbl@evargs{,% <- don't delete this comma
1024   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1025   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1026   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1027   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1028   beforestart=0,language=2}
1029 \ifx\NewHook\undefined\else
1030   \def\bbl@tempa#1=#2@@{\NewHook{babel/#1}}
1031   \bbl@foreach\bbl@evargs{\bbl@tempa#1@@}
1032 \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 `\babelensure{<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.

```

1033 \bbl@trace{Defining babelensure}
1034 \newcommand\babelensure[2][{}]{%
1035   \AddBabelHook{babel-ensure}{afterextras}{%
1036     \ifcase\bbl@select@type
1037       \bbl@c1{e}%
1038     \fi}%
1039   \begingroup
1040     \let\bbl@ens@include\@empty
1041     \let\bbl@ens@exclude\@empty
1042     \def\bbl@ens@fontenc{\relax}%
1043     \def\bbl@tempb##1{%
1044       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1045     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1046     \def\bbl@tempb##1=##2\@{\@namedef\bbl@ens@##1}{##2}}%
1047     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1048     \def\bbl@tempc{\bbl@ensure}%
1049     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1050       \expandafter{\bbl@ens@include}}%
1051     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1052       \expandafter{\bbl@ens@exclude}}%
1053     \toks@\expandafter{\bbl@tempc}%
1054     \bbl@exp{%
1055   \endgroup
1056   \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1057 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1058   \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1059     \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1060       \edef##1{\noexpand\bbl@nocaption
1061         {\bbl@stripslash##1}{\language\name\bbl@stripslash##1}}%
1062       \fi
1063       \ifx##1\@empty\else
1064         \in{##1}{#2}%
1065         \ifin@ \else
1066           \bbl@ifunset{\bbl@ensure@\language\name}%
1067             {\bbl@exp{%
1068               \\DeclareRobustCommand\<bbl@ensure@\language\name>[1]{%
1069                 \\foreignlanguage{\language\name}%
1070                 {\ifx\relax#3\else
1071                   \\fontencoding{#3}\\selectfont
1072                   \fi
1073                   #####1}}}%
1074             }%
1075           \toks@\expandafter{##1}%
1076           \edef##1{%
1077             \bbl@csarg\noexpand{ensure@\language\name}%
1078             {\the\toks@}}%
1079           \fi
1080           \expandafter\bbl@tempb
1081         \fi}%
1082   \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1083   \def\bbl@tempa##1{% elt for include list
1084     \ifx##1\@empty\else
1085       \bbl@csarg\in{ensure@\language\name\expandafter}\expandafter{##1}%
1086       \ifin@ \else
1087         \bbl@tempb##1\@empty
1088       \fi
1089       \expandafter\bbl@tempa
1090     \fi}%
1091   \bbl@tempa#1\@empty}
1092 \def\bbl@captionslist{%
1093   \prefacename\refname\abstractname\bibname\chaptername\appendixname

```

```

1094 \contentsname\listfigurename\listtablename\indexname\figurename
1095 \tablename\partname\enclname\ccname\headtoname\pagename\seename
1096 \alsiname\proofname\glossaryname}

```

## 7.4 Setting up language files

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

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

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

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

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

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

Finally we check `\originalTeX`.

```

1097 \bbl@trace{Macros for setting language files up}
1098 \def\bbl@ldfinit{%
1099   \let\bbl@screset\@empty
1100   \let\BabelStrings\bbl@opt@string
1101   \let\BabelOptions\@empty
1102   \let\BabelLanguages\relax
1103   \ifx\originalTeX\@undefined
1104     \let\originalTeX\@empty
1105   \else
1106     \originalTeX
1107   \fi}
1108 \def\LdfInit#1#2{%
1109   \chardef\atcatcode=\catcode`\@
1110   \catcode`\@=11\relax
1111   \chardef\eqcatcode=\catcode`\=
1112   \catcode`\==12\relax
1113   \expandafter\if\expandafter\@backslashchar
1114     \expandafter\@car\string#2\@nil
1115     \ifx#2\@undefined\else
1116       \ldf@quit{#1}%
1117     \fi
1118   \else
1119     \expandafter\ifx\csname#2\endcsname\relax\else
1120       \ldf@quit{#1}%
1121     \fi
1122   \fi
1123   \bbl@ldfinit}

```

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

```

1124 \def\ldf@quit#1{%
1125   \expandafter\main@language\expandafter{#1}%
1126   \catcode`\@=\atcatcode \let\atcatcode\relax
1127   \catcode`\==\eqcatcode \let\eqcatcode\relax
1128   \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.

```

1129 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1130   \bbl@afterlang
1131   \let\bbl@afterlang\relax
1132   \let\BabelModifiers\relax
1133   \let\bbl@screset\relax}%
1134 \def\ldf@finish#1{%
1135   \loadlocalcfg{#1}%
1136   \bbl@afterldf{#1}%
1137   \expandafter\main@language\expandafter{#1}%
1138   \catcode`\@=\atcatcode \let\atcatcode\relax
1139   \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 L<sup>A</sup>T<sub>E</sub>X.

```

1140 \@onlypreamble\LdfInit
1141 \@onlypreamble\ldf@quit
1142 \@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.

```

1143 \def\main@language#1{%
1144   \def\bbl@main@language{#1}%
1145   \let\language\name\bbl@main@language % TODO. Set locale name
1146   \bbl@id@assign
1147   \bbl@patterns{\language}

```

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

```

1148 \def\bbl@beforestart{%
1149   \def\@nolanerr##1{%
1150     \bbl@warning{Undefined language '##1' in aux.\@Reported}}%
1151   \bbl@usehooks{beforestart}{}%
1152   \global\let\bbl@beforestart\relax}
1153 \AtBeginDocument{%
1154   {\@nameuse{bbl@beforestart}}% Group!
1155   \if@filesw
1156     \providecommand\babel@aux[2]{}%
1157     \immediate\write\@mainaux{%
1158       \string\providecommand\string\babel@aux[2]{}%
1159       \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1160     }%
1161   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1162   \ifbbl@single % must go after the line above.
1163     \renewcommand\selectlanguage[1]{}%
1164     \renewcommand\foreignlanguage[2]{#2}%
1165     \global\let\babel@aux\@gobbletwo % Also as flag
1166   \fi
1167   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1168 \def\select@language@x#1{%
1169   \ifcase\bbl@select@type
1170     \bbl@ifsamestring\language\name{#1}{\select@language{#1}}%
1171   \else
1172     \select@language{#1}%
1173   \fi}

```



## 7.5 Shorthands

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

```
1174 \bbl@trace{Shorthands}
1175 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1176   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1177   \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\@makeother#1}}%
1178   \ifx\nfss@catcodes\undefined\else % TODO - same for above
1179     \begingroup
1180       \catcode`#1\active
1181       \nfss@catcodes
1182       \ifnum\catcode`#1=\active
1183         \endgroup
1184         \bbl@add\nfss@catcodes{\@makeother#1}%
1185       \else
1186         \endgroup
1187       \fi
1188   \fi}
```

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

```
1189 \def\bbl@remove@special#1{%
1190   \begingroup
1191     \def\x##1##2{\ifnum`#1=``##2\noexpand\empty
1192       \else\noexpand##1\noexpand##2\fi}%
1193     \def\do{\x\do}%
1194     \def\@makeother{\x\@makeother}%
1195   \edef\x{\endgroup
1196     \def\noexpand\dospecials{\dospecials}%
1197     \expandafter\ifx\csname @sanitize\endcsname\relax\else
1198       \def\noexpand\@sanitize{\@sanitize}%
1199     \fi}%
1200   \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).

```
1201 \def\bbl@active@def#1#2#3#4{%
1202   \@namedef{#3#1}{%
1203     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1204       \bbl@afterelse\bbl@sh@select#2#1{#3#arg#1}{#4#1}%
1205     \else
1206       \bbl@afterfi\csname#2@sh@#1\endcsname
```

```
1207 \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.

```
1208 \long\@namedef{#3@arg#1}##1{%
1209 \expandafter\ifx\csname#2@sh@#1@\string##1@endcsname\relax
1210 \bbl@afterelse\csname#4#1@endcsname##1%
1211 \else
1212 \bbl@afterfi\csname#2@sh@#1@\string##1@endcsname
1213 \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.

```
1214 \def\initiate@active@char#1{%
1215 \bbl@ifunset{active@char\string#1}%
1216 {\bbl@withactive
1217 {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1218 {}}
```

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).

```
1219 \def\@initiate@active@char#1#2#3{%
1220 \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1221 \ifx#1\@undefined
1222 \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1223 \else
1224 \bbl@csarg\let{oridef@#2}#1%
1225 \bbl@csarg\edef{oridef@#2}{%
1226 \let\noexpand#1%
1227 \expandafter\noexpand\csname bbl@oridef@@#2@endcsname}%
1228 \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*").

```
1229 \ifx#1#3\relax
1230 \expandafter\let\csname normal@char#2@endcsname#3%
1231 \else
1232 \bbl@info{Making #2 an active character}%
1233 \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1234 \@namedef{normal@char#2}{%
1235 \textormath{#3}\csname bbl@oridef@@#2@endcsname}}%
1236 \else
1237 \@namedef{normal@char#2}{#3}%
1238 \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).

```
1239 \bbl@restoreactive{#2}%
1240 \AtBeginDocument{%
1241 \catcode`#2\active
1242 \if@files@w
1243 \immediate\write\@mainaux{\catcode`\string#2\active}%
1244 \fi}%
1245 \expandafter\bbl@add@special\csname#2@endcsname
1246 \catcode`#2\active
1247 \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⟩`).

```

1248 \let\bbl@tempa\@firstoftwo
1249 \if\string^#2%
1250   \def\bbl@tempa{\noexpand\textormath}%
1251 \else
1252   \ifx\bbl@mathnormal\@undefined\else
1253     \let\bbl@tempa\bbl@mathnormal
1254   \fi
1255 \fi
1256 \expandafter\edef\csname active@char#2\endcsname{%
1257   \bbl@tempa
1258     {\noexpand\if@safe@actives
1259       \noexpand\expandafter
1260         \expandafter\noexpand\csname normal@char#2\endcsname
1261       \noexpand\else
1262         \noexpand\expandafter
1263         \expandafter\noexpand\csname bbl@doactive#2\endcsname
1264       \noexpand\fi}%
1265   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1266 \bbl@csarg\edef{doactive#2}{%
1267   \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!).

```

1268 \bbl@csarg\edef{active@#2}{%
1269   \noexpand\active@prefix\noexpand#1%
1270   \expandafter\noexpand\csname active@char#2\endcsname}%
1271 \bbl@csarg\edef{normal@#2}{%
1272   \noexpand\active@prefix\noexpand#1%
1273   \expandafter\noexpand\csname normal@char#2\endcsname}%
1274 \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.

```

1275 \bbl@active@def#2\user@group{user@active}{language@active}%
1276 \bbl@active@def#2\language@group{language@active}{system@active}%
1277 \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.

```

1278 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1279   {\expandafter\noexpand\csname normal@char#2\endcsname}%
1280 \expandafter\edef\csname\user@group @sh#2@\string\protect@\endcsname
1281   {\expandafter\noexpand\csname user@active#2\endcsname}%

```

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

```

1282 \if\string'#2%
1283   \let\prim@s\bbl@prim@s

```

```

1284 \let\active@math@prime#1%
1285 \fi
1286 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}

```

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

```

1287 <<{*More package options}>> ≡
1288 \DeclareOption{math=active}{}
1289 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1290 <</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.

```

1291 \@ifpackagewith{babel}{KeepShorthandsActive}%
1292 {\let\bbl@restoreactive\@gobble}%
1293 {\def\bbl@restoreactive#1{%
1294   \bbl@exp{%
1295     \\\AfterBabelLanguage\\CurrentOption
1296     {\catcode`#1=\the\catcode`#1\relax}%
1297     \\\AtEndOfPackage
1298     {\catcode`#1=\the\catcode`#1\relax}}}%
1299   \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.

```

1300 \def\bbl@sh@select#1#2{%
1301   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1302     \bbl@afterelse\bbl@scndcs
1303   \else
1304     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1305   \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.

```

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

```

```

1328         \protect#1%
1329         \fi
1330         \expandafter\expandafter\expandafter\@gobble
1331         \fi
1332         \fi}}
1333 \endgroup

```

`\if@safe@actives` In some circumstances it is necessary to be able to set the catcode of a shorthand to ‘other’ on the fly (for example, if things like `\char"AA` are expected). For this purpose the switch `@safe@actives` is available. The setting of this switch should be checked in the first level expansion of `\active@char<char>`. When this expansion mode is active, something like `"13"13` becomes `"12"12` (in other words, they are `\string`’ed). This contrasts with `\protected@edef`, where catcodes are left unchanged.

```

1334 \newif\if@safe@actives
1335 \@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.

```

1336 \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`.

```

1337 \chardef\bbl@activated\z@
1338 \def\bbl@activate#1{%
1339   \chardef\bbl@activated\@ne
1340   \bbl@withactive{\expandafter\let\expandafter}#1%
1341   \csname bbl@active@\string#1\endcsname}
1342 \def\bbl@deactivate#1{%
1343   \chardef\bbl@activated\tw@
1344   \bbl@withactive{\expandafter\let\expandafter}#1%
1345   \csname bbl@normal@\string#1\endcsname}

```

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

```

\bbl@scndcs
1346 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1347 \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.

```

1348 \def\babel@texpdf#1#2#3#4{%
1349   \ifx\texorpdfstring\undefined
1350     \textormath{#1}{#3}%
1351   \else
1352     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1353     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1354   \fi}
1355 %
1356 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1357 \def\@decl@short#1#2#3\@nil#4{%
1358   \def\bbl@tempa{#3}%
1359   \ifx\bbl@tempa\empty
1360     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs

```

```

1361 \bbl@ifunset{#1@sh@\string#2@}{}%
1362 {\def\bbl@tempa{#4}%
1363 \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1364 \else
1365 \bbl@info
1366 {Redefining #1 shorthand \string#2\\%
1367 in language \CurrentOption}%
1368 \fi}%
1369 \@namedef{#1@sh@\string#2@}{#4}%
1370 \else
1371 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1372 \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1373 {\def\bbl@tempa{#4}%
1374 \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1375 \else
1376 \bbl@info
1377 {Redefining #1 shorthand \string#2\string#3\\%
1378 in language \CurrentOption}%
1379 \fi}%
1380 \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1381 \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.

```

1382 \def\textormath{%
1383 \ifmmode
1384 \expandafter\@secondoftwo
1385 \else
1386 \expandafter\@firstoftwo
1387 \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’.

```

1388 \def\user@group{user}
1389 \def\language@group{english} % TODO. I don't like defaults
1390 \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.

```

1391 \def\usesshorthands{%
1392 \@ifstar\bbl@usesesh@s{\bbl@usesesh@x{}}
1393 \def\bbl@usesesh@s#1{%
1394 \bbl@usesesh@x
1395 {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1396 {#1}}
1397 \def\bbl@usesesh@x#1#2{%
1398 \bbl@ifshorthand{#2}%
1399 {\def\user@group{user}%
1400 \initiate@active@char{#2}%
1401 #1%
1402 \bbl@activate{#2}}%
1403 {\bbl@error
1404 {I can't declare a shorthand turned off (\string#2)}
1405 {Sorry, but you can't use shorthands which have been\\%
1406 turned off in the package options}}}

```

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

```

1407 \def\user@language@group{user@\language@group}
1408 \def\bbl@set@user@generic#1#2{%
1409   \bbl@ifunset{user@generic@active#1}%
1410   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1411     \bbl@active@def#1\user@group{user@generic@active}{language@active}%
1412     \expandafter\edef\csname#2@sh@#1@\endcsname{%
1413       \expandafter\noexpand\csname normal@char#1\endcsname}%
1414     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1415       \expandafter\noexpand\csname user@active#1\endcsname}}%
1416   \@empty}
1417 \newcommand\defineshorthand[3][user]{%
1418   \edef\bbl@tempa{\zap@space#1 \@empty}%
1419   \bbl@for\bbl@tempb\bbl@tempa{%
1420     \if*\expandafter\@car\bbl@tempb\@nil
1421       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1422       \@expandtwoargs
1423       \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1424     \fi
1425     \declare@shorthand{\bbl@tempb}{#2}{#3}}

```

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

```

1426 \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`.

```

1427 \def\aliasshorthand#1#2{%
1428   \bbl@ifshorthand{#2}%
1429   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1430     \ifx\document\@notprerr
1431       \@notshorthand{#2}%
1432     \else
1433       \initiate@active@char{#2}%
1434       \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1435       \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1436       \bbl@activate{#2}%
1437     \fi
1438   \fi}%
1439   {\bbl@error
1440     {Cannot declare a shorthand turned off (\string#2)}
1441     {Sorry, but you cannot use shorthands which have been\\
1442       turned off in the package options}}}

```

`\@notshorthand`

```

1443 \def\@notshorthand#1{%
1444   \bbl@error{%
1445     The character '\string #1' should be made a shorthand character;\\%
1446     add the command \string\usesshorthands\string{#1\string} to
1447     the preamble.\\%
1448     I will ignore your instruction}%
1449   {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.

```

1450 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1451 \DeclareRobustCommand*\shorthandoff{%
1452   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1453 \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.

```

1454 \def\bb@switch@sh#1#2{%
1455   \ifx#2\@nnil\else
1456     \bb@ifunset{bb@active@\string#2}%
1457     {\bb@error
1458       {I can't switch '\string#2' on or off--not a shorthand}%
1459       {This character is not a shorthand. Maybe you made\%
1460         a typing mistake? I will ignore your instruction.}}%
1461     {\ifcase#1%   off, on, off*
1462       \catcode`#2\relax
1463     \or
1464       \catcode`#2\active
1465       \bb@ifunset{bb@shdef@\string#2}%
1466       {%
1467         {\bb@withactive{\expandafter\let\expandafter}#2%
1468           \csname bb@shdef@\string#2\endcsname
1469           \bb@csarg\let{shdef@\string#2}\relax}%
1470         \ifcase\bb@activated\or
1471           \bb@activate{#2}%
1472         \else
1473           \bb@deactivate{#2}%
1474         \fi
1475       \or
1476         \bb@ifunset{bb@shdef@\string#2}%
1477         {\bb@withactive{\bb@csarg\let{shdef@\string#2}}#2}%
1478         {%
1479           \csname bb@oricat@\string#2\endcsname
1480           \csname bb@oridef@\string#2\endcsname
1481         \fi}%
1482     \bb@afterfi\bb@switch@sh#1%
1483   \fi}

```

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

```

1484 \def\babelshorthand{\active@prefix\babelshorthand\bb@putsh}
1485 \def\bb@putsh#1{%
1486   \bb@ifunset{bb@active@\string#1}%
1487   {\bb@putsh@i#1\@empty\@nnil}%
1488   {\csname bb@active@\string#1\endcsname}}
1489 \def\bb@putsh@i#1#2\@nnil{%
1490   \csname\language@group @sh@\string#1@%
1491     \ifx\@empty#2\else\string#2\fi\endcsname}
1492 %
1493 \ifx\bb@opt@shorthands\@nnil\else
1494   \let\bb@s@initiate@active@char\initiate@active@char
1495   \def\initiate@active@char#1{%
1496     \bb@ifshorthand{#1}{\bb@s@initiate@active@char{#1}}{}}
1497   \let\bb@s@switch@sh\bb@switch@sh
1498   \def\bb@switch@sh#1#2{%
1499     \ifx#2\@nnil\else
1500       \bb@afterfi
1501       \bb@ifshorthand{#2}{\bb@s@switch@sh#1{#2}}{\bb@switch@sh#1}%
1502     \fi}
1503   \let\bb@s@activate\bb@activate
1504   \def\bb@activate#1{%
1505     \bb@ifshorthand{#1}{\bb@s@activate{#1}}{}}
1506   \let\bb@s@deactivate\bb@deactivate
1507   \def\bb@deactivate#1{%
1508     \bb@ifshorthand{#1}{\bb@s@deactivate{#1}}{}}
1509 \fi

```



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

```
1510 \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  
`\bbl@pr@m@s` 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.

```
1511 \def\bbl@prim@s{%
1512   \prime\futurelet\@let@token\bbl@pr@m@s}
1513 \def\bbl@if@primes#1#2{%
1514   \ifx#1\@let@token
1515     \expandafter\@firstoftwo
1516   \else\ifx#2\@let@token
1517     \bbl@afterelse\expandafter\@firstoftwo
1518   \else
1519     \bbl@afterfi\expandafter\@secondoftwo
1520   \fi\fi}
1521 \begingroup
1522   \catcode`\^=7 \catcode`\*=\active \lccode`\^=\^
1523   \catcode`\'=12 \catcode`\"=\active \lccode`\"="\'
1524   \lowercase{%
1525     \gdef\bbl@pr@m@s{%
1526       \bbl@if@primes""%
1527       \pr@@@s
1528       {\bbl@if@primes*\^pr@@@t\egroup}}
1529 \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).

```
1530 \initiate@active@char{~}
1531 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1532 \bbl@activate{~}
```

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

```
1533 \expandafter\def\csname OT1dqpos\endcsname{127}
1534 \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

```
1535 \ifx\f@encoding\undefined
1536   \def\f@encoding{OT1}
1537 \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.

```
1538 \bbl@trace{Language attributes}
1539 \newcommand\languageattribute[2]{%
1540   \def\bbl@tempc{#1}%
1541   \bbl@fixname\bbl@tempc
1542   \bbl@iflanguage\bbl@tempc{%
1543     \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.

```

1544 \ifx\bbl@known@attrs\@undefined
1545 \in@false
1546 \else
1547 \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
1548 \fi
1549 \ifin@
1550 \bbl@warning{%
1551 You have more than once selected the attribute '##1'\%
1552 for language #1. Reported}%
1553 \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.

```

1554 \bbl@exp{%
1555 \\\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
1556 \edef\bbl@tempa{\bbl@tempc-##1}%
1557 \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1558 {\csname\bbl@tempc @attr##1\endcsname}%
1559 {\@attrerr{\bbl@tempc}{##1}}%
1560 \fi}}
1561 \onlypreamble\languageattribute

```

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

```

1562 \newcommand*{\@attrerr}[2]{%
1563 \bbl@error
1564 {The attribute #2 is unknown for language #1.}%
1565 {Your command will be ignored, type <return> to proceed}}

```

`\bbl@declare@attribute` 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}`.

```

1566 \def\bbl@declare@attribute#1#2#3{%
1567 \bbl@xin@{,#2,}{,\BabelModifiers,}%
1568 \ifin@
1569 \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1570 \fi
1571 \bbl@add@list\bbl@attributes{#1-#2}%
1572 \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.

```

1573 \def\bbl@ifattributeset#1#2#3#4{%
1574 \ifx\bbl@known@attrs\@undefined
1575 \in@false
1576 \else
1577 \bbl@xin@{,#1-#2,}{,\bbl@known@attrs,}%
1578 \fi
1579 \ifin@
1580 \bbl@afterelse#3%
1581 \else
1582 \bbl@afterfi#4%
1583 \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.

```

1584 \def\bbl@ifknown@ttrib#1#2{%
1585   \let\bbl@tempa\@secondoftwo
1586   \bbl@loopx\bbl@tempb{#2}{%
1587     \expandafter\in\expandafter{\expandafter,\bbl@tempb,}{, #1,}%
1588     \ifin@
1589       \let\bbl@tempa\@firstoftwo
1590     \else
1591       \fi}%
1592   \bbl@tempa}

```

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

```

1593 \def\bbl@clear@ttribs{%
1594   \ifx\bbl@attributes\@undefined\else
1595     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1596       \expandafter\bbl@clear@ttrib\bbl@tempa.
1597     }%
1598     \let\bbl@attributes\@undefined
1599   \fi}
1600 \def\bbl@clear@ttrib#1-#2.{%
1601   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1602 \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`

```

1603 \bbl@trace{Macros for saving definitions}
1604 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

1605 \newcount\babel@savecnt
1606 \babel@beginsave

```

`\babel@save` The macro `\babel@save⟨csname⟩` saves the current meaning of the control sequence `⟨csname⟩` 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. To avoid messing saved definitions up, they are saved only the very first time.

```

1607 \def\babel@save#1{%
1608   \def\bbl@tempa{, #1,}% Clumsy, for Plain
1609   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1610     \expandafter{\expandafter,\bbl@savextras,}%
1611     \expandafter\in\bbl@tempa
1612     \ifin@\else
1613       \bbl@add\bbl@savextras{, #1,}%
1614       \bbl@carg\let{\babel@number\babel@savecnt}#1\relax
1615       \toks@\expandafter{\originalTeX\let#1=}
1616       \bbl@exp{%
1617         \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%

```

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

```

1618 \advance\babel@savecnt\@ne
1619 \fi}
1620 \def\babel@savevariable#1{%
1621 \toks@\expandafter{\originalTeX #1}%
1622 \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.

```

1623 \def\bbl@frenchspacing{%
1624 \ifnum\the\scode`\.=\@m
1625 \let\bbl@nonfrenchspacing\relax
1626 \else
1627 \frenchspacing
1628 \let\bbl@nonfrenchspacing\nonfrenchspacing
1629 \fi}
1630 \let\bbl@nonfrenchspacing\nonfrenchspacing
1631 \let\bbl@elt\relax
1632 \edef\bbl@fs@chars{%
1633 \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1634 \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1635 \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1636 \def\bbl@pre@fs{%
1637 \def\bbl@elt##1##2##3{\scode`##1=\the\scode`##1\relax}%
1638 \edef\bbl@save@sfcodes{\bbl@fs@chars}}%
1639 \def\bbl@post@fs{%
1640 \bbl@save@sfcodes
1641 \edef\bbl@tempa{\bbl@cl{frspc}}%
1642 \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1643 \if u\bbl@tempa % do nothing
1644 \else\if n\bbl@tempa % non french
1645 \def\bbl@elt##1##2##3{%
1646 \ifnum\scode`##1=##2\relax
1647 \babel@savevariable{\scode`##1}%
1648 \scode`##1=##3\relax
1649 \fi}%
1650 \bbl@fs@chars
1651 \else\if y\bbl@tempa % french
1652 \def\bbl@elt##1##2##3{%
1653 \ifnum\scode`##1=##3\relax
1654 \babel@savevariable{\scode`##1}%
1655 \scode`##1=##2\relax
1656 \fi}%
1657 \bbl@fs@chars
1658 \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.

```

1659 \bbl@trace{Short tags}
1660 \def\babeltags#1{%
1661 \edef\bbl@tempa{\zap@space#1 \@empty}%
1662 \def\bbl@tempb##1=##2\@{ }%
1663 \edef\bbl@tempc{%
1664 \noexpand\newcommand
1665 \expandafter\noexpand\csname ##1\endcsname{%
1666 \noexpand\protect
1667 \expandafter\noexpand\csname other language*\endcsname{##2}}

```

```

1668 \noexpand\newcommand
1669 \expandafter\noexpand\csname text##1\endcsname{%
1670 \noexpand\foreignlanguage{##2}}%
1671 \bbl@tempc}%
1672 \bbl@for\bbl@tempa\bbl@tempa{%
1673 \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.

```

1674 \bbl@trace{Hyphens}
1675 \@onlypreamble\babelhyphenation
1676 \AtEndOfPackage{%
1677 \newcommand\babelhyphenation[2][\@empty]{%
1678 \ifx\bbl@hyphenation@relax
1679 \let\bbl@hyphenation@\@empty
1680 \fi
1681 \ifx\bbl@hyphlist\@empty\else
1682 \bbl@warning{%
1683 You must not intermingle \string\selectlanguage\space and\\%
1684 \string\babelhyphenation\space or some exceptions will not\\%
1685 be taken into account. Reported}%
1686 \fi
1687 \ifx\@empty#1%
1688 \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1689 \else
1690 \bbl@vforeach{#1}{%
1691 \def\bbl@tempa{##1}%
1692 \bbl@fixname\bbl@tempa
1693 \bbl@iflanguage\bbl@tempa{%
1694 \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1695 \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1696 }{%
1697 {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1698 #2}}}%
1699 \fi}}

```

`\bbl@allowhyphens` This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak \hskip Opt plus Opt`<sup>32</sup>.

```

1700 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1701 \def\bbl@t@one{T1}
1702 \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`.

```

1703 \newcommand\babellnullhyphen{\char\hyphenchar\font}
1704 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1705 \def\bbl@hyphen{%
1706 \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
1707 \def\bbl@hyphen@i#1#2{%
1708 \bbl@ifunset{bbl@hy#1#2\@empty}%
1709 {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1710 {\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.

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

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.

```
1711 \def\bbl@usehyphen#1{%
1712   \leavevmode
1713   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1714   \nobreak\hskip\z@skip}
1715 \def\bbl@@usehyphen#1{%
1716   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
1717 \def\bbl@hyphenchar{%
1718   \ifnum\hyphenchar\font=\m@ne
1719     \babe\lnullhyphen
1720   \else
1721     \char\hyphenchar\font
1722   \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.

```
1723 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1724 \def\bbl@hy@@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1725 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1726 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1727 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}{}}
1728 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1729 \def\bbl@hy@repeat{%
1730   \bbl@usehyphen{%
1731     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1732 \def\bbl@hy@@repeat{%
1733   \bbl@usehyphen{%
1734     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1735 \def\bbl@hy@empty{\hskip\z@skip}
1736 \def\bbl@hy@@empty{\discretionary{}{}{}}
```

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

```
1737 \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.

```
1738 \bbl@trace{Multiencoding strings}
1739 \def\bbl@tglobal#1{\global\let#1#1}
```

The second one. We need to patch \@ucllist, 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 \@ucllist 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).

```
1740 \@ifpackagewith{babel}{nocase}%
1741   {\let\bbl@patchuclc\relax}%
```

```

1742 {\def\bbbl@patchuclc{%
1743 \global\let\bbbl@patchuclc\relax
1744 \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbbl@uclc}}%
1745 \gdef\bbbl@uclc##1{%
1746 \let\bbbl@encoded\bbbl@encoded@uclc
1747 \bbbl@ifunset{\language @bbbl@uclc}% and resumes it
1748 {##1}%
1749 {\let\bbbl@tempa##1\relax % Used by LANG@bbbl@uclc
1750 \csname\language @bbbl@uclc\endcsname}%
1751 {\bbbl@tolower\@empty}{\bbbl@toupper\@empty}}%
1752 \gdef\bbbl@tolower{\csname\language @bbbl@lc\endcsname}%
1753 \gdef\bbbl@toupper{\csname\language @bbbl@uc\endcsname}}
1754 <<More package options>> ≡
1755 \DeclareOption{nocase}{}
1756 <</More package options>>

```

The following package options control the behavior of \SetString.

```

1757 <<More package options>> ≡
1758 \let\bbbl@opt@strings\@nnil % accept strings=value
1759 \DeclareOption{strings}{\def\bbbl@opt@strings{\BabelStringsDefault}}
1760 \DeclareOption{strings=encoded}{\let\bbbl@opt@strings\relax}
1761 \def\BabelStringsDefault{generic}
1762 <</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.

```

1763 \@onlypreamble\StartBabelCommands
1764 \def\StartBabelCommands{%
1765 \begingroup
1766 \@tempcnta="7F
1767 \def\bbbl@tempa{%
1768 \ifnum\@tempcnta>"FF\else
1769 \catcode\@tempcnta=11
1770 \advance\@tempcnta\@ne
1771 \expandafter\bbbl@tempa
1772 \fi}%
1773 \bbbl@tempa
1774 <<Macros local to BabelCommands>>
1775 \def\bbbl@provstring##1##2{%
1776 \providecommand##1{##2}%
1777 \bbbl@tglobal##1}%
1778 \global\let\bbbl@scafter\@empty
1779 \let\StartBabelCommands\bbbl@startcmds
1780 \ifx\BabelLanguages\relax
1781 \let\BabelLanguages\CurrentOption
1782 \fi
1783 \begingroup
1784 \let\bbbl@screset\@nnil % local flag - disable 1st stopcommands
1785 \StartBabelCommands}
1786 \def\bbbl@startcmds{%
1787 \ifx\bbbl@screset\@nnil\else
1788 \bbbl@usehooks{stopcommands}{}%
1789 \fi
1790 \endgroup
1791 \begingroup
1792 \@ifstar
1793 {\ifx\bbbl@opt@strings\@nnil
1794 \let\bbbl@opt@strings\BabelStringsDefault
1795 \fi
1796 \bbbl@startcmds@i}%
1797 \bbbl@startcmds@i}

```

```

1798 \def\bbl@startcmds@i#1#2{%
1799   \edef\bbl@L{\zap@space#1 \@empty}%
1800   \edef\bbl@G{\zap@space#2 \@empty}%
1801   \bbl@startcmds@ii}
1802 \let\bbl@startcommands\StartBabelCommands

```

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

Select the behavior of \SetString. There are two main cases, depending of if there is an optional argument: without it and strings=encoded, strings are defined always; otherwise, they are set only if they are still undefined (ie, fallback values). With labelled blocks and strings=encoded, define the strings, but with another value, define strings only if the current label or font encoding is the value of strings; otherwise (ie, no strings or a block whose label is not in strings=) do nothing.

We presume the current block is not loaded, and therefore set (above) a couple of default values to gobble the arguments. Then, these macros are redefined if necessary according to several parameters.

```

1803 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1804   \let\SetString\@gobbletwo
1805   \let\bbl@stringdef\@gobbletwo
1806   \let\AfterBabelCommands\@gobble
1807   \ifx\@empty#1%
1808     \def\bbl@sc@label{generic}%
1809     \def\bbl@encstring##1##2{%
1810       \ProvideTextCommandDefault##1{##2}%
1811       \bbl@tglobal##1%
1812       \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%
1813       \let\bbl@sctest\in@true
1814     \else
1815       \let\bbl@sc@charset\space % <- zapped below
1816       \let\bbl@sc@fontenc\space % <- " "
1817       \def\bbl@tempa##1=##2\@nil{%
1818         \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1819       \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1820       \def\bbl@tempa##1 ##2{% space -> comma
1821         ##1%
1822         \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1823       \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1824       \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1825       \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1826       \def\bbl@encstring##1##2{%
1827         \bbl@foreach\bbl@sc@fontenc{%
1828           \bbl@ifunset{T@####1}%
1829           {}%
1830           {\ProvideTextCommand##1{####1}{##2}%
1831             \bbl@tglobal##1%
1832             \expandafter
1833             \bbl@tglobal\csname####1\string##1\endcsname}}}%
1834       \def\bbl@sctest{%
1835         \bbl@xin@{\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1836       \fi
1837       \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
1838       \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
1839         \let\AfterBabelCommands\bbl@aftercmds
1840         \let\SetString\bbl@setstring
1841         \let\bbl@stringdef\bbl@encstring
1842       \else % ie, strings=value
1843         \bbl@sctest
1844       \fin@
1845       \let\AfterBabelCommands\bbl@aftercmds
1846       \let\SetString\bbl@setstring
1847       \let\bbl@stringdef\bbl@provstring
1848       \fi\fi\fi
1849       \bbl@scswitch
1850       \ifx\bbl@G\@empty

```



```

1851 \def\SetString##1##2{%
1852     \bbl@error{Missing group for string \string##1}%
1853     {You must assign strings to some category, typically\\%
1854     captions or extras, but you set none}}%
1855 \fi
1856 \ifx\@empty#1%
1857     \bbl@usehooks{defaultcommands}{}%
1858 \else
1859     \@expandtwoargs
1860     \bbl@usehooks{encodedcommands}{\bbl@sc@charset}\bbl@sc@fontenc}%
1861 \fi}

```

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

```

1862 \def\bbl@forlang#1#2{%
1863     \bbl@for#1\bbl@L{%
1864         \bbl@xin@{,#1},{,\BabelLanguages,}%
1865         \ifin#2\relax\fi}}
1866 \def\bbl@scswitch{%
1867     \bbl@forlang\bbl@tempa{%
1868         \ifx\bbl@G\@empty\else
1869             \ifx\SetString\@gobbletwo\else
1870                 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1871                 \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
1872                 \ifin\else
1873                     \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1874                     \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1875                 \fi
1876             \fi
1877         \fi}}
1878 \AtEndOfPackage{%
1879     \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\#2}}}%
1880     \let\bbl@scswitch\relax}
1881 \@onlypreamble\EndBabelCommands
1882 \def\EndBabelCommands{%
1883     \bbl@usehooks{stopcommands}{}%
1884     \endgroup
1885     \endgroup
1886     \bbl@scafter}
1887 \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.

```

1888 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1889     \bbl@forlang\bbl@tempa{%
1890         \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1891         \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1892         {\bbl@exp{%
1893             \global\bbbl@add\<\bbl@G\bbl@tempa>\bbbl@scset\#1\<\bbl@LC>}}}%
1894         {}}%
1895     \def\BabelString{#2}%
1896     \bbl@usehooks{stringprocess}{}%

```

```

1897 \expandafter\bb1@stringdef
1898 \csname\bb1@LC\expandafter\endcsname\expandafter{\BabelString}}

```

Now, some additional stuff to be used when encoded strings are used. Captions then include `\bb1@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`.

```

1899 \ifx\bb1@opt@strings\relax
1900 \def\bb1@scset#1#2{\def#1{\bb1@encoded#2}}
1901 \bb1@patchuclc
1902 \let\bb1@encoded\relax
1903 \def\bb1@encoded@uclc#1{%
1904   \@inmathwarn#1%
1905   \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1906     \expandafter\ifx\csname ?\string#1\endcsname\relax
1907       \TextSymbolUnavailable#1%
1908     \else
1909       \csname ?\string#1\endcsname
1910     \fi
1911   \else
1912     \csname\cf@encoding\string#1\endcsname
1913   \fi}
1914 \else
1915 \def\bb1@scset#1#2{\def#1{#2}}
1916 \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.

```

1917 << *Macros local to BabelCommands >> ≡
1918 \def\SetStringLoop##1##2{%
1919   \def\bb1@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1920   \count@\z@
1921   \bb1@loop\bb1@tempa{##2}{% empty items and spaces are ok
1922     \advance\count@\@ne
1923     \toks@\expandafter{\bb1@tempa}%
1924     \bb1@exp{%
1925       \SetString\bb1@templ{\romannumeral\count@}{\the\toks@}%
1926       \count@=\the\count@\relax}}}%
1927 <</Macros local to BabelCommands >>

```

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

```

1928 \def\bb1@aftercmds#1{%
1929   \toks@\expandafter{\bb1@scafter#1}%
1930   \xdef\bb1@scafter{\the\toks@}}

```

**Case mapping** The command `\SetCase` provides a way to change the behavior of `\MakeUppercase` and `\MakeLowercase`. `\bb1@tempa` is set by the patched `\@uclclist` to the parsing command.

```

1931 << *Macros local to BabelCommands >> ≡
1932 \newcommand\SetCase[3][{}]{%
1933   \bb1@patchuclc
1934   \bb1@forlang\bb1@tempa{%
1935     \bb1@carg\bb1@encstring{\bb1@tempa @bb1@uclc}{\bb1@tempa##1}%
1936     \bb1@carg\bb1@encstring{\bb1@tempa @bb1@uc}{##2}%
1937     \bb1@carg\bb1@encstring{\bb1@tempa @bb1@lc}{##3}}}%
1938 <</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.

```

1939 << *Macros local to BabelCommands >> ≡
1940 \newcommand\SetHyphenMap[1]{%

```

```

1941 \bbl@forlang\bbl@tempa{%
1942 \expandafter\bbl@stringdef
1943 \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1944 <</Macros local to BabelCommands>>

```

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

```

1945 \newcommand\BabelLower[2]{% one to one.
1946 \ifnum\lccode#1=#2\else
1947 \babel@savevariable{\lccode#1}%
1948 \lccode#1=#2\relax
1949 \fi}
1950 \newcommand\BabelLowerMM[4]{% many-to-many
1951 \@tempcnta=#1\relax
1952 \@tempcntb=#4\relax
1953 \def\bbl@tempa{%
1954 \ifnum\@tempcnta>#2\else
1955 \expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1956 \advance\@tempcnta#3\relax
1957 \advance\@tempcntb#3\relax
1958 \expandafter\bbl@tempa
1959 \fi}%
1960 \bbl@tempa}
1961 \newcommand\BabelLowerMO[4]{% many-to-one
1962 \@tempcnta=#1\relax
1963 \def\bbl@tempa{%
1964 \ifnum\@tempcnta>#2\else
1965 \expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1966 \advance\@tempcnta#3
1967 \expandafter\bbl@tempa
1968 \fi}%
1969 \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1970 <<{*More package options}>> ≡
1971 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1972 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap@ne}
1973 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1974 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@}
1975 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1976 <</More package options>>

```

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

```

1977 \AtEndOfPackage{%
1978 \ifx\bbl@opt@hyphenmap\undefined
1979 \bbl@xin@{,}{\bbl@language@opts}%
1980 \chardef\bbl@opt@hyphenmap\ifin4\else\@ne\fi
1981 \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.

```

1982 \newcommand\setlocalecaption{% TODO. Catch typos.
1983 \ifstar\bbl@setcaption@s\bbl@setcaption@x}
1984 \def\bbl@setcaption@x#1#2#3{% language caption-name string
1985 \bbl@trim@def\bbl@tempa{#2}%
1986 \bbl@xin@{.template}{\bbl@tempa}%
1987 \ifin@
1988 \bbl@ini@captions@template{#3}{#1}%
1989 \else
1990 \edef\bbl@tempd{%
1991 \expandafter\expandafter\expandafter
1992 \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1993 \bbl@xin@
1994 {\expandafter\string\csname #2name\endcsname}%

```

```

1995     {\bbl@tempd}%
1996 \ifin@ % Renew caption
1997     \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
1998     \ifin@
1999         \bbl@exp{%
2000             \\bbl@ifsamestring{\bbl@tempa}{\language}%
2001             {\bbl@scset\<#2name>\<#1#2name>}%
2002             {}}%
2003     \else % Old way converts to new way
2004         \bbl@ifunset{#1#2name}%
2005         {\bbl@exp{%
2006             \\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2007             \\bbl@ifsamestring{\bbl@tempa}{\language}%
2008             {\def\<#2name>{\<#1#2name>}}%
2009             {}}}%
2010         {}}%
2011     \fi
2012 \else
2013     \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2014     \ifin@ % New way
2015         \bbl@exp{%
2016             \\bbl@add\<captions#1>{\bbl@scset\<#2name>\<#1#2name>}%
2017             \\bbl@ifsamestring{\bbl@tempa}{\language}%
2018             {\bbl@scset\<#2name>\<#1#2name>}%
2019             {}}%
2020     \else % Old way, but defined in the new way
2021         \bbl@exp{%
2022             \\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2023             \\bbl@ifsamestring{\bbl@tempa}{\language}%
2024             {\def\<#2name>{\<#1#2name>}}%
2025             {}}%
2026     \fi%
2027 \fi
2028 \@namedef{#1#2name}{#3}%
2029 \toks@\expandafter{\bbl@captionslist}%
2030 \bbl@exp{\in@{\<#2name>}{\the\toks@}}%
2031 \ifin@ \else
2032     \bbl@exp{\bbl@add\bbl@captionslist{\<#2name>}}%
2033     \bbl@tglobal\bbl@captionslist
2034 \fi
2035 \fi}
2036 % \def\bbl@setcaption@s#1#2#3{} % TODO. Not yet implemented (w/o 'name')

```

## 7.11 Macros common to a number of languages

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

```

2037 \bbl@trace{Macros related to glyphs}
2038 \def\set@low@box#1{\setbox\tw\hbox{,}\setbox\z@ \hbox{#1}%
2039     \dimen\z@ \ht\z@ \advance\dimen\z@ -\ht\tw@%
2040     \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.

```

2041 \def\save@sf@q#1{\leavevmode
2042     \begingroup
2043     \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2044     \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.

```
2045 \ProvideTextCommand{\quotedblbase}{OT1}{%
2046   \save@sf@q{\set@low@box{\textquotedblright\}}%
2047   \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.

```
2048 \ProvideTextCommandDefault{\quotedblbase}{%
2049   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2050 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2051   \save@sf@q{\set@low@box{\textquoteright\}}%
2052   \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.

```
2053 \ProvideTextCommandDefault{\quotesinglbase}{%
2054   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2055 \ProvideTextCommand{\guillemetleft}{OT1}{%
2056   \ifmmode
2057     \ll
2058   \else
2059     \save@sf@q{\nobreak
2060       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2061     \fi}
2062 \ProvideTextCommand{\guillemetright}{OT1}{%
2063   \ifmmode
2064     \gg
2065   \else
2066     \save@sf@q{\nobreak
2067       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2068     \fi}
2069 \ProvideTextCommand{\guillemotleft}{OT1}{%
2070   \ifmmode
2071     \ll
2072   \else
2073     \save@sf@q{\nobreak
2074       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2075     \fi}
2076 \ProvideTextCommand{\guillemotright}{OT1}{%
2077   \ifmmode
2078     \gg
2079   \else
2080     \save@sf@q{\nobreak
2081       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2082     \fi}
```

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

```
2083 \ProvideTextCommandDefault{\guillemetleft}{%
2084   \UseTextSymbol{OT1}{\guillemetleft}}
2085 \ProvideTextCommandDefault{\guillemetright}{%
2086   \UseTextSymbol{OT1}{\guillemetright}}
2087 \ProvideTextCommandDefault{\guillemotleft}{%
2088   \UseTextSymbol{OT1}{\guillemotleft}}
2089 \ProvideTextCommandDefault{\guillemotright}{%
2090   \UseTextSymbol{OT1}{\guillemotright}}
```

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

`\guilsinglright`

```

2091 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2092   \ifmmode
2093     <%
2094   \else
2095     \save@sf@q{\nobreak
2096       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2097   \fi}
2098 \ProvideTextCommand{\guilsinglright}{OT1}{%
2099   \ifmmode
2100     >%
2101   \else
2102     \save@sf@q{\nobreak
2103       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2104   \fi}

```

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

```

2105 \ProvideTextCommandDefault{\guilsinglleft}{%
2106   \UseTextSymbol{OT1}{\guilsinglleft}}
2107 \ProvideTextCommandDefault{\guilsinglright}{%
2108   \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.

```

2109 \DeclareTextCommand{\ij}{OT1}{%
2110   i\kern-0.02em\bbl@allowhyphens j}
2111 \DeclareTextCommand{\IJ}{OT1}{%
2112   I\kern-0.02em\bbl@allowhyphens J}
2113 \DeclareTextCommand{\ij}{T1}{\char188}
2114 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2115 \ProvideTextCommandDefault{\ij}{%
2116   \UseTextSymbol{OT1}{\ij}}
2117 \ProvideTextCommandDefault{\IJ}{%
2118   \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).

```

2119 \def\crrtic@{\hrule height0.1ex width0.3em}
2120 \def\crttic@{\hrule height0.1ex width0.33em}
2121 \def\ddj@{%
2122   \setbox0\hbox{d}\dimen@=\ht0
2123   \advance\dimen@1ex
2124   \dimen@.45\dimen@
2125   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2126   \advance\dimen@ii.5ex
2127   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2128 \def\DDJ@{%
2129   \setbox0\hbox{D}\dimen@=.55\ht0
2130   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2131   \advance\dimen@ii.15ex % correction for the dash position
2132   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2133   \dimen\thr@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2134   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2135 %
2136 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2137 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```



## 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).

```
2174 \def\umlauthigh{%
2175   \def\bbl@umlauta##1{\leavevmode\bgroup%
2176     \accent\csname\fontencoding dqpos\endcsname
2177     ##1\bbl@allowhyphens\egroup}%
2178   \let\bbl@umlaute\bbl@umlauta}
2179 \def\umlautlow{%
2180   \def\bbl@umlauta{\protect\lower@umlaut}}
2181 \def\umlautelow{%
2182   \def\bbl@umlaute{\protect\lower@umlaut}}
2183 \umlauthigh
```

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

```
2184 \expandafter\ifx\csname U@D\endcsname\relax
2185   \csname newdimen\endcsname\U@D
2186 \fi
```

The following code fools T<sub>E</sub>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.

```
2187 \def\lower@umlaut#1{%
2188   \leavevmode\bgroup
2189   \U@D 1ex%
2190   {\setbox\z@\hbox{%
2191     \char\csname\fontencoding dqpos\endcsname}%
2192     \dimen@ -.45ex\advance\dimen@\ht\z@
2193     \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2194   \accent\csname\fontencoding dqpos\endcsname
2195   \fontdimen5\font\U@D #1%
2196   \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).

```
2197 \AtBeginDocument{%
2198   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2199   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2200   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2201   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{i}}%
2202   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
2203   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
2204   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
2205   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2206   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2207   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
2208   \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.

```
2209 \ifx\l@english\@undefined
2210   \chardef\l@english\z@
2211 \fi
2212 % The following is used to cancel rules in ini files (see Amharic).
2213 \ifx\l@unhyphenated\@undefined
2214   \newlanguage\l@unhyphenated
2215 \fi
```

## 7.13 Layout

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

```
2216 \bbl@trace{Bidi layout}
2217 \providecommand\IfBabelLayout[3]{#3}%
2218 \newcommand\BabelPatchSection[1]{%
2219   \@ifundefined{#1}{}{%
2220     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2221     \@namedef{#1}{%
2222       \ifstar{\bbl@presec@#1}%
2223       {\@dblarg{\bbl@presec@x{#1}}}}}%
2224 \def\bbl@presec@x#1[#2]#3{%
2225   \bbl@exp{%
2226     \\\select@language@x{\bbl@main@language}%
2227     \\\bbl@cs{sspre@#1}%
2228     \\\bbl@cs{ss@#1}%
2229     [\\foreignlanguage{\language}{\unexpanded{#2}}]%
2230     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2231     \\\select@language@x{\language}}}%
2232 \def\bbl@presec@#1#2{%
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     \\\select@language@x{\language}}}%
2239 \IfBabelLayout{sectioning}%
2240   {\BabelPatchSection{part}%
2241    \BabelPatchSection{chapter}%
2242    \BabelPatchSection{section}%
2243    \BabelPatchSection{subsection}%
2244    \BabelPatchSection{subsubsection}%
2245    \BabelPatchSection{paragraph}%
2246    \BabelPatchSection{subparagraph}%
2247    \def\babel@toc#1{%
2248      \select@language@x{\bbl@main@language}}}%
2249 \IfBabelLayout{captions}%
2250   {\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.

```
2251 \bbl@trace{Input engine specific macros}
2252 \ifcase\bbl@engine
2253   \input txtbabel.def
2254 \or
2255   \input luababel.def
2256 \or
2257   \input xebabel.def
2258 \fi
2259 \providecommand\babelfont{%
```

```

2260 \bbl@error
2261 {This macro is available only in LuaLaTeX and XeLaTeX.}%
2262 {Consider switching to these engines.}}
2263 \providecommand\babelprehyphenation{%
2264 \bbl@error
2265 {This macro is available only in LuaLaTeX.}%
2266 {Consider switching to that engine.}}
2267 \ifx\babelposthyphenation\@undefined
2268 \let\babelposthyphenation\babelprehyphenation
2269 \let\babelpatterns\babelprehyphenation
2270 \let\babelcharproperty\babelprehyphenation
2271 \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.

```

2272 \bbl@trace{Creating languages and reading ini files}
2273 \let\bbl@extend@ini\@gobble
2274 \newcommand\babelprovide[2][]{%
2275 \let\bbl@savelangname\language
2276 \edef\bbl@savelocaleid{\the\localeid}%
2277 % Set name and locale id
2278 \edef\language{#2}%
2279 \bbl@id@assign
2280 % Initialize keys
2281 \bbl@vforeach{captions,date,import,main,script,language,%
2282 hyphenrules,linebreaking,justification,mapfont,maparabic,%
2283 mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2284 Alph,labels,labels*,calendar,date}%
2285 {\bbl@csarg\let{KVP@##1}\@nnil}%
2286 \global\let\bbl@release@transforms\@empty
2287 \let\bbl@calendars\@empty
2288 \global\let\bbl@inidata\@empty
2289 \global\let\bbl@extend@ini\@gobble
2290 \gdef\bbl@key@list{;}%
2291 \bbl@forkv{#1}{%
2292 \in@{/}{##1}%
2293 \ifin@
2294 \global\let\bbl@extend@ini\bbl@extend@ini@aux
2295 \bbl@renewinikey##1\@{##2}%
2296 \else
2297 \bbl@csarg\ifx{KVP@##1}\@nnil\else
2298 \bbl@error
2299 {Unknown key '##1' in \string\babelprovide}%
2300 {See the manual for valid keys}%
2301 \fi
2302 \bbl@csarg\def{KVP@##1}{##2}%
2303 \fi}%
2304 \chardef\bbl@howloaded=0:none; 1:ldf without ini; 2:ini
2305 \bbl@ifunset{date#2}\z@{\bbl@ifunset{\bbl@llevel@#2}\ne\tw@}%
2306 % == init ==
2307 \ifx\bbl@screset\@undefined
2308 \bbl@ldfinit
2309 \fi
2310 % == date (as option) ==
2311 % \ifx\bbl@KVP@date\@nnil\else
2312 % \fi
2313 % ==
2314 \let\bbl@lbkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2315 \ifcase\bbl@howloaded
2316 \let\bbl@lbkflag\@empty % new

```

```

2317 \else
2318 \ifx\bb1@KVP@hyphenrules\@nnil\else
2319 \let\bb1@lbfkflag\@empty
2320 \fi
2321 \ifx\bb1@KVP@import\@nnil\else
2322 \let\bb1@lbfkflag\@empty
2323 \fi
2324 \fi
2325 % == import, captions ==
2326 \ifx\bb1@KVP@import\@nnil\else
2327 \bb1@exp{\bb1@ifblank{\bb1@KVP@import}}%
2328 {\ifx\bb1@initload\relax
2329 \begin{group}
2330 \def\BabelBeforeIni##1##2{\gdef\bb1@KVP@import{##1}\endinput}%
2331 \bb1@input@texini{##2}%
2332 \end{group}
2333 \else
2334 \xdef\bb1@KVP@import{\bb1@initload}%
2335 \fi}%
2336 {}%
2337 \let\bb1@KVP@date\@empty
2338 \fi
2339 \let\bb1@KVP@captions@\bb1@KVP@captions % TODO. A dirty hack
2340 \ifx\bb1@KVP@captions\@nnil
2341 \let\bb1@KVP@captions\bb1@KVP@import
2342 \fi
2343 % ==
2344 \ifx\bb1@KVP@transforms\@nnil\else
2345 \bb1@replace\bb1@KVP@transforms{ }{,}%
2346 \fi
2347 % == Load ini ==
2348 \ifcase\bb1@howloaded
2349 \bb1@provide@new{##2}%
2350 \else
2351 \bb1@ifblank{##1}%
2352 {}% With \bb1@load@basic below
2353 {\bb1@provide@renew{##2}}%
2354 \fi
2355 % Post tasks
2356 % -----
2357 % == subsequent calls after the first provide for a locale ==
2358 \ifx\bb1@inidata\@empty\else
2359 \bb1@extend@ini{##2}%
2360 \fi
2361 % == ensure captions ==
2362 \ifx\bb1@KVP@captions\@nnil\else
2363 \bb1@ifunset{\bb1@extracaps@##2}%
2364 {\bb1@exp{\bb1@babelensure[exclude=\today]{##2}}}%
2365 {\bb1@exp{\bb1@babelensure[exclude=\today,
2366 include=\bb1@extracaps@##2]{##2}}}%
2367 \bb1@ifunset{\bb1@ensure@language}%
2368 {\bb1@exp{%
2369 \\\DeclareRobustCommand\<\bb1@ensure@language>[1]{%
2370 \\\foreignlanguage{\language}%
2371 {###1}}}%
2372 {}%
2373 \bb1@exp{%
2374 \\\bb1@tglobal\<\bb1@ensure@language>%
2375 \\\bb1@tglobal\<\bb1@ensure@language\space>}%
2376 \fi
2377 % ==
2378 % At this point all parameters are defined if 'import'. Now we
2379 % execute some code depending on them. But what about if nothing was

```

```

2380 % imported? We just set the basic parameters, but still loading the
2381 % whole ini file.
2382 \bbl@load@basic{#2}%
2383 % == script, language ==
2384 % Override the values from ini or defines them
2385 \ifx\bbl@KVP@script\@nnil\else
2386   \bbl@csarg\edef{sname#2}{\bbl@KVP@script}%
2387 \fi
2388 \ifx\bbl@KVP@language\@nnil\else
2389   \bbl@csarg\edef{lname#2}{\bbl@KVP@language}%
2390 \fi
2391 \ifcase\bbl@engine\or
2392   \bbl@ifunset{\bbl@chrng@\languagename}{}%
2393   {\directlua{
2394     Babel.set_chranges_b('\bbl@cl{sbcpr}', '\bbl@cl{chrng}') }}%
2395 \fi
2396 % == onchar ==
2397 \ifx\bbl@KVP@onchar\@nnil\else
2398   \bbl@luahyphenate
2399   \bbl@exp{%
2400     \\\AddToHook{env/document/before}{\select@language{#2}}}%
2401   \directlua{
2402     if Babel.locale_mapped == nil then
2403       Babel.locale_mapped = true
2404       Babel.linebreaking.add_before(Babel.locale_map, 1)
2405       Babel.loc_to_scr = {}
2406       Babel.chr_to_loc = Babel.chr_to_loc or {}
2407     end
2408     Babel.locale_props[\the\localeid].letters = false
2409   }%
2410   \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
2411   \ifin@
2412     \directlua{
2413       Babel.locale_props[\the\localeid].letters = true
2414     }%
2415   \fi
2416   \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2417   \ifin@
2418     \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
2419       \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
2420     \fi
2421     \bbl@exp{\\bbl@add\\bbl@starthyphens
2422       {\bbl@patterns@lua{\languagename}}}%
2423     % TODO - error/warning if no script
2424     \directlua{
2425       if Babel.script_blocks['\bbl@cl{sbcpr}'] then
2426         Babel.loc_to_scr[\the\localeid] =
2427           Babel.script_blocks['\bbl@cl{sbcpr}']
2428         Babel.locale_props[\the\localeid].lc = \the\localeid\space
2429         Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space
2430       end
2431     }%
2432   \fi
2433   \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2434   \ifin@
2435     \bbl@ifunset{\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}}%
2436     \bbl@ifunset{\bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}}%
2437     \directlua{
2438       if Babel.script_blocks['\bbl@cl{sbcpr}'] then
2439         Babel.loc_to_scr[\the\localeid] =
2440           Babel.script_blocks['\bbl@cl{sbcpr}']
2441       end}%
2442     \ifx\bbl@mapselect\@undefined % TODO. almost the same as mapfont

```

```

2443 \AtBeginDocument{%
2444 \bbl@patchfont{\bbl@mapselect}}%
2445 {\selectfont}}%
2446 \def\bbl@mapselect{%
2447 \let\bbl@mapselect\relax
2448 \edef\bbl@prefontid{\fontid\font}}%
2449 \def\bbl@mapdir##1{%
2450 {\def\language{##1}%
2451 \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2452 \bbl@switchfont
2453 \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2454 \directlua{
2455 Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
2456 ['\bbl@prefontid'] = \fontid\font\space}%
2457 \fi}}%
2458 \fi
2459 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
2460 \fi
2461 % TODO - catch non-valid values
2462 \fi
2463 % == mapfont ==
2464 % For bidi texts, to switch the font based on direction
2465 \ifx\bbl@KVP@mapfont\@nnil\else
2466 \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}%
2467 {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\%
2468 mapfont. Use 'direction'.}%
2469 {See the manual for details.}}%
2470 \bbl@ifunset{bbl@lsys@\language}{\bbl@provide@lsys{\language}}}%
2471 \bbl@ifunset{bbl@wdir@\language}{\bbl@provide@dirs{\language}}}%
2472 \ifx\bbl@mapselect\@undefined % TODO. See onchar.
2473 \AtBeginDocument{%
2474 \bbl@patchfont{\bbl@mapselect}}%
2475 {\selectfont}}%
2476 \def\bbl@mapselect{%
2477 \let\bbl@mapselect\relax
2478 \edef\bbl@prefontid{\fontid\font}}%
2479 \def\bbl@mapdir##1{%
2480 {\def\language{##1}%
2481 \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2482 \bbl@switchfont
2483 \directlua{Babel.fontmap
2484 [\the\csname bbl@wdir@##1\endcsname]%
2485 [\bbl@prefontid]=\fontid\font}}}%
2486 \fi
2487 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
2488 \fi
2489 % == Line breaking: intraspace, intrapenalty ==
2490 % For CJK, East Asian, Southeast Asian, if interspace in ini
2491 \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2492 \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2493 \fi
2494 \bbl@provide@intraspace
2495 % == Line breaking: CJK quotes ==
2496 \ifcase\bbl@engine\or
2497 \bbl@xin@{/c}{\bbl@c1{lnbrk}}%
2498 \ifin@
2499 \bbl@ifunset{bbl@quote@\language}}%
2500 {\directlua{
2501 Babel.locale_props[\the\localeid].cjk_quotes = {}
2502 local cs = 'op'
2503 for c in string.utfvalues(
2504 [[\csname bbl@quote@\language\endcsname]]) do
2505 if Babel.cjk_characters[c].c == 'qu' then

```

```

2506         Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2507     end
2508     cs = ( cs == 'op') and 'cl' or 'op'
2509 end
2510 }}%
2511 \fi
2512 \fi
2513 % == Line breaking: justification ==
2514 \ifx\bbbl@KVP@justification\@nnil\else
2515     \let\bbbl@KVP@linebreaking\bbbl@KVP@justification
2516 \fi
2517 \ifx\bbbl@KVP@linebreaking\@nnil\else
2518     \bbbl@xin@{\, \bbbl@KVP@linebreaking,}%
2519     {,elongated,kashida,cjk,padding,unhyphenated,}%
2520 \ifin@
2521     \bbbl@csarg\xdef
2522     {\lnbrk@\language}\{ \expandafter \@car \bbbl@KVP@linebreaking \@nil}%
2523 \fi
2524 \fi
2525 \bbbl@xin@{/e}{ \bbbl@cl{\lnbrk}}%
2526 \ifin@\else \bbbl@xin@{/k}{ \bbbl@cl{\lnbrk}} \fi
2527 \ifin@\bbbl@arabicjust \fi
2528 \bbbl@xin@{/p}{ \bbbl@cl{\lnbrk}}%
2529 \ifin@\AtBeginDocument{\@nameuse{\bbbl@tibetanjust}} \fi
2530 % == Line breaking: hyphenate.other.(locale|script) ==
2531 \ifx\bbbl@lbcflag\@empty
2532     \bbbl@ifunset{\bbbl@hyotl@\language}{}%
2533     {\bbbl@csarg\bbbl@replace{\hyotl@\language}{ }{ },}%
2534     \bbbl@startcommands*{\language}{}%
2535     \bbbl@csarg\bbbl@foreach{\hyotl@\language}{%
2536         \ifcase\bbbl@engine
2537             \ifnum##1<257
2538                 \SetHyphenMap{\BabelLower{##1}{##1}}%
2539             \fi
2540             \else
2541                 \SetHyphenMap{\BabelLower{##1}{##1}}%
2542             \fi}%
2543     \bbbl@endcommands}%
2544 \bbbl@ifunset{\bbbl@hyots@\language}{}%
2545 {\bbbl@csarg\bbbl@replace{\hyots@\language}{ }{ },}%
2546 \bbbl@csarg\bbbl@foreach{\hyots@\language}{%
2547     \ifcase\bbbl@engine
2548         \ifnum##1<257
2549             \global\lccode##1=##1\relax
2550         \fi
2551         \else
2552             \global\lccode##1=##1\relax
2553         \fi}%
2554 \fi
2555 % == Counters: maparabic ==
2556 % Native digits, if provided in ini (TeX level, xe and lua)
2557 \ifcase\bbbl@engine\else
2558     \bbbl@ifunset{\bbbl@dgnat@\language}{}%
2559     {\expandafter\ifx\csname \bbbl@dgnat@\language\endcsname\@empty\else
2560         \expandafter\expandafter\expandafter
2561         \bbbl@setdigits\csname \bbbl@dgnat@\language\endcsname
2562         \ifx\bbbl@KVP@maparabic\@nnil\else
2563             \ifx\bbbl@latinarabic\@undefined
2564                 \expandafter\let\expandafter\@arabic
2565                 \csname \bbbl@counter@\language\endcsname
2566             \else % ie, if layout=counters, which redefines \@arabic
2567                 \expandafter\let\expandafter\bbbl@latinarabic
2568                 \csname \bbbl@counter@\language\endcsname

```

```

2569         \fi
2570     \fi
2571     \fi}%
2572 \fi
2573 % == Counters: mapdigits ==
2574 % > luababel.def
2575 % == Counters: alph, Alph ==
2576 \ifx\bbbl@KVP@alph\@nnil\else
2577     \bbbl@exp{%
2578         \\bbbl@add\<bbbl@preextras@\languagename>{%
2579             \\babel@save\\@alph
2580             \let\\@alph\<bbbl@cntr@\bbbl@KVP@alph @\languagename>}}%
2581 \fi
2582 \ifx\bbbl@KVP@Alph\@nnil\else
2583     \bbbl@exp{%
2584         \\bbbl@add\<bbbl@preextras@\languagename>{%
2585             \\babel@save\\@Alph
2586             \let\\@Alph\<bbbl@cntr@\bbbl@KVP@Alph @\languagename>}}%
2587 \fi
2588 % == Calendars ==
2589 \ifx\bbbl@KVP@calendar\@nnil
2590     \edef\bbbl@KVP@calendar{\bbbl@cl{calpr}}}%
2591 \fi
2592 \def\bbbl@tempe##1 ##2\@{% Get first calendar
2593     \def\bbbl@tempa{##1}}%
2594     \bbbl@exp{\\bbbl@tempe\bbbl@KVP@calendar\space\\@}%
2595 \def\bbbl@tempe##1.##2.##3\@{%
2596     \def\bbbl@tempc{##1}%
2597     \def\bbbl@tempb{##2}}%
2598 \expandafter\bbbl@tempe\bbbl@tempa..\@
2599 \bbbl@csarg\edef{calpr@\languagename}{%
2600     \ifx\bbbl@tempc\@empty\else
2601         calendar=\bbbl@tempc
2602     \fi
2603     \ifx\bbbl@tempb\@empty\else
2604         ,variant=\bbbl@tempb
2605     \fi}%
2606 % == engine specific extensions ==
2607 % Defined in XXXbabel.def
2608 \bbbl@provide@extra{#2}%
2609 % == require.babel in ini ==
2610 % To load or reload the babel-*.tex, if require.babel in ini
2611 \ifx\bbbl@beforestart\relax\else % But not in doc aux or body
2612     \bbbl@ifunset{bbbl@rqtex@\languagename}{}%
2613     {\expandafter\ifx\csname bbbl@rqtex@\languagename\endcsname\@empty\else
2614         \let\BabelBeforeIni\@gobbletwo
2615         \chardef\atcatcode=\catcode`\@
2616         \catcode`\@=11\relax
2617         \bbbl@input@texini{\bbbl@cs{rqtex@\languagename}}%
2618         \catcode`\@=\atcatcode
2619         \let\atcatcode\relax
2620         \global\bbbl@csarg\let{rqtex@\languagename}\relax
2621     \fi}%
2622 \bbbl@foreach\bbbl@calendars{%
2623     \bbbl@ifunset{bbbl@ca##1}{%
2624         \chardef\atcatcode=\catcode`\@
2625         \catcode`\@=11\relax
2626         \InputIfFileExists{babel-ca-##1.tex}{}}%
2627         \catcode`\@=\atcatcode
2628         \let\atcatcode\relax}%
2629     {}}%
2630 \fi
2631 % == frenchspacing ==

```

```

2632 \ifcase\bbbl@howloaded\in@true\else\in@false\fi
2633 \ifin@ \else\bbbl@xin@{typography/frenchspacing}{\bbbl@key@list}\fi
2634 \ifin@
2635   \bbbl@extras@wrap{\bbbl@pre@fs}%
2636   {\bbbl@pre@fs}%
2637   {\bbbl@post@fs}%
2638 \fi
2639 % == transforms ==
2640 % > luababel.def
2641 % == main ==
2642 \ifx\bbbl@KVP@main\@nnil % Restore only if not 'main'
2643   \let\languagename\bbbl@savelangname
2644   \chardef\localeid\bbbl@savelocaleid\relax
2645 \fi}

```

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

```

2646 \def\bbbl@provide@new#1{%
2647   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2648   \@namedef{extras#1}{}%
2649   \@namedef{noextras#1}{}%
2650   \bbbl@startcommands*{#1}{captions}%
2651   \ifx\bbbl@KVP@captions\@nnil % and also if import, implicit
2652     \def\bbbl@tempb##1{% elt for \bbbl@captionslist
2653       \ifx##1\@empty\else
2654         \bbbl@exp{%
2655           \\\SetString\\##1{%
2656             \\\bbbl@nocaption{\bbbl@stripslash##1}{#1\bbbl@stripslash##1}}}%
2657           \expandafter\bbbl@tempb
2658         \fi}%
2659     \expandafter\bbbl@tempb\bbbl@captionslist\@empty
2660   \else
2661     \ifx\bbbl@initoload\relax
2662       \bbbl@read@ini{\bbbl@KVP@captions}2% % Here letters cat = 11
2663     \else
2664       \bbbl@read@ini{\bbbl@initoload}2% % Same
2665     \fi
2666   \fi
2667   \StartBabelCommands*{#1}{date}%
2668   \ifx\bbbl@KVP@date\@nnil
2669     \bbbl@exp{%
2670       \\\SetString\\today{\bbbl@nocaption{today}{#1today}}}%
2671   \else
2672     \bbbl@savetoday
2673     \bbbl@savestate
2674   \fi
2675   \bbbl@endcommands
2676   \bbbl@load@basic{#1}%
2677   % == hyphenmins == (only if new)
2678   \bbbl@exp{%
2679     \gdef\<#1hyphenmins>{%
2680       {\bbbl@ifunset{\bbbl@lfthm@#1}{2}{\bbbl@cs{lfthm@#1}}}%
2681       {\bbbl@ifunset{\bbbl@rgthm@#1}{3}{\bbbl@cs{rgthm@#1}}}}}%
2682   % == hyphenrules (also in renew) ==
2683   \bbbl@provide@hyphens{#1}%
2684   \ifx\bbbl@KVP@main\@nnil\else
2685     \expandafter\main@language\expandafter{#1}%
2686   \fi}
2687 %
2688 \def\bbbl@provide@renew#1{%
2689   \ifx\bbbl@KVP@captions\@nnil\else
2690     \StartBabelCommands*{#1}{captions}%
2691     \bbbl@read@ini{\bbbl@KVP@captions}2% % Here all letters cat = 11

```



```

2692 \EndBabelCommands
2693 \fi
2694 \ifx\bbbl@KVP@date\@nnil\else
2695 \StartBabelCommands*{#1}{date}%
2696 \bbbl@savetoday
2697 \bbbl@savedate
2698 \EndBabelCommands
2699 \fi
2700 % == hyphenrules (also in new) ==
2701 \ifx\bbbl@lbfkflag\@empty
2702 \bbbl@provide@hyphens{#1}%
2703 \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.)

```

2704 \def\bbbl@load@basic#1{%
2705 \ifcase\bbbl@howloaded\or\or
2706 \ifcase\csname bbl@llevel\language\endcsname
2707 \bbbl@csarg\let\lname\language\relax
2708 \fi
2709 \fi
2710 \bbbl@ifunset{bbbl@lname@#1}%
2711 {\def\BabelBeforeIni##1##2{%
2712 \begingroup
2713 \let\bbbl@ini@captions@aux\@gobbletwo
2714 \def\bbbl@inidate ####1.####2.####3.####4\relax ####5####6}%
2715 \bbbl@read@ini{##1}1%
2716 \ifx\bbbl@initoload\relax\endinput\fi
2717 \endgroup}%
2718 \begingroup % boxed, to avoid extra spaces:
2719 \ifx\bbbl@initoload\relax
2720 \bbbl@input@texini{#1}%
2721 \else
2722 \setbox\z@\hbox{\BabelBeforeIni{\bbbl@initoload}}}%
2723 \fi
2724 \endgroup}%
2725 {}%

```

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

```

2726 \def\bbbl@provide@hyphens#1{%
2727 \@tempcnta\m@ne % a flag
2728 \ifx\bbbl@KVP@hyphenrules\@nnil\else
2729 \bbbl@replace\bbbl@KVP@hyphenrules{ }{,}%
2730 \bbbl@foreach\bbbl@KVP@hyphenrules{%
2731 \ifnum\@tempcnta=\m@ne % if not yet found
2732 \bbbl@ifsamestring{##1}{+}%
2733 {\bbbl@carg\addlanguage{l@##1}}%
2734 }%
2735 \bbbl@ifunset{l@##1}% After a possible +
2736 {}%
2737 {\@tempcnta\@nameuse{l@##1}}%
2738 \fi}%
2739 \ifnum\@tempcnta=\m@ne
2740 \bbbl@warning{%
2741 Requested 'hyphenrules=' for '\language' not found.\%
2742 Using the default value. Reported}%
2743 \fi
2744 \fi
2745 \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found
2746 \ifx\bbbl@KVP@captions@\@nnil % TODO. Hackish. See above.
2747 \bbbl@ifunset{bbbl@hyphr@#1}% use value in ini, if exists
2748 {\bbbl@exp{\bbbl@ifblank{\bbbl@cs{hyphr@#1}}}%

```

```

2749      {}%
2750      {\bbl@ifunset{1@\bbl@cl{hyphr}}}%
2751      {}%          if hyphenrules found:
2752      {\@tempcnta\@nameuse{1@\bbl@cl{hyphr}}}}}%
2753      \fi
2754    \fi
2755    \bbl@ifunset{1@#1}%
2756      {\ifnum\@tempcnta=\m@ne
2757        \bbl@carg\adddialect{1@#1}\language
2758      \else
2759        \bbl@carg\adddialect{1@#1}\@tempcnta
2760      \fi}%
2761      {\ifnum\@tempcnta=\m@ne\else
2762        \global\bbl@carg\chardef{1@#1}\@tempcnta
2763      \fi}}

```

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

```

2764 \def\bbl@input@texini#1{%
2765   \bbl@bsphack
2766   \bbl@exp{%
2767     \catcode`\%%=14 \catcode`\%%=0
2768     \catcode`\%{=1 \catcode`\%}=2
2769     \lowercase{\InputIfFileExists{babel-#1.tex}{}}}%
2770     \catcode`\%%=\the\catcode`\%\relax
2771     \catcode`\%{=\the\catcode`\%\relax
2772     \catcode`\%{=\the\catcode`\%\relax
2773     \catcode`\%{=\the\catcode`\%\relax}%
2774   \bbl@esphack}

```

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

```

2775 \def\bbl@iniline#1\bbl@iniline{%
2776   \@ifnextchar[\bbl@iniset{\@ifnextchar\bbl@iniskip\bbl@inistore}#1\@@}% ]
2777 \def\bbl@iniset[#1]#2\@@{\def\bbl@section{#1}}
2778 \def\bbl@iniskip#1\@@{}%      if starts with ;
2779 \def\bbl@inistore#1=#2\@@{}%   full (default)
2780 \bbl@trim\def\bbl@tempa{#1}%
2781 \bbl@trim\toks@{#2}%
2782 \bbl@xin@;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2783 \ifin@ \else
2784   \bbl@xin@{,identification/include.}%
2785   {,\bbl@section/\bbl@tempa}%
2786   \ifin@\edef\bbl@required@inis{\the\toks@}\fi
2787   \bbl@exp{%
2788     \\\g@addto@macro\\bbl@inidata{%
2789       \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2790   \fi}
2791 \def\bbl@inistore@min#1=#2\@@{}% minimal (maybe set in \bbl@read@ini)
2792 \bbl@trim\def\bbl@tempa{#1}%
2793 \bbl@trim\toks@{#2}%
2794 \bbl@xin@{.identification.}{.\bbl@section.}%
2795 \ifin@
2796   \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2797     \\\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2798   \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.

```

2799 \def\bbl@loop@ini{%

```

```

2800 \loop
2801 \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2802 \endlinechar\m@ne
2803 \read\bbl@readstream to \bbl@line
2804 \endlinechar\^^M
2805 \ifx\bbl@line\@empty\else
2806 \expandafter\bbl@iniline\bbl@line\bbl@iniline
2807 \fi
2808 \repeat}
2809 \ifx\bbl@readstream\@undefined
2810 \csname newread\endcsname\bbl@readstream
2811 \fi
2812 \def\bbl@read@ini#1#2{%
2813 \global\let\bbl@extend@ini\@gobble
2814 \openin\bbl@readstream=babel-#1.ini
2815 \ifeof\bbl@readstream
2816 \bbl@error
2817 {There is no ini file for the requested language\\%
2818 (#1: \language). Perhaps you misspelled it or your\\%
2819 installation is not complete.}%
2820 {Fix the name or reinstall babel.}%
2821 \else
2822 % == Store ini data in \bbl@inidata ==
2823 \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2824 \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2825 \bbl@info{Importing
2826 \ifcase#2font and identification \or basic \fi
2827 data for \language\\%
2828 from babel-#1.ini. Reported}%
2829 \ifnum#2=\z@
2830 \global\let\bbl@inidata\@empty
2831 \let\bbl@inistore\bbl@inistore@min % Remember it's local
2832 \fi
2833 \def\bbl@section{identification}%
2834 \let\bbl@required@inis\@empty
2835 \bbl@exp{\\bbl@inistore tag.ini=#1\\@@}%
2836 \bbl@inistore load.level=#2\\@@
2837 \bbl@loop@ini
2838 \ifx\bbl@required@inis\@empty\else
2839 \bbl@replace\bbl@required@inis{ },}%
2840 \bbl@foreach\bbl@required@inis{%
2841 \openin\bbl@readstream=##1.ini
2842 \bbl@loop@ini}%
2843 \fi
2844 % == Process stored data ==
2845 \bbl@csarg\xdef{lini@\language}{#1}%
2846 \bbl@read@ini@aux
2847 % == 'Export' data ==
2848 \bbl@ini@exports{#2}%
2849 \global\bbl@csarg\let{inidata@\language}\bbl@inidata
2850 \global\let\bbl@inidata\@empty
2851 \bbl@exp{\\bbl@add@list\\bbl@ini@loaded{\language}}%
2852 \bbl@tglobal\bbl@ini@loaded
2853 \fi}
2854 \def\bbl@read@ini@aux{%
2855 \let\bbl@savestrings\@empty
2856 \let\bbl@savetoday\@empty
2857 \let\bbl@savestate\@empty
2858 \def\bbl@elt##1##2##3{%
2859 \def\bbl@section{##1}%
2860 \in@{=date.}{=##1}% Find a better place
2861 \ifin@
2862 \bbl@ifunset{bbl@inikv@##1}%

```

```

2863     {\bbl@ini@calendar{##1}}%
2864     }%
2865     \fi
2866     \in@{=identification/extension.}{=##1/##2}%
2867     \ifin@
2868         \bbl@ini@extension{##2}%
2869     \fi
2870     \bbl@ifunset{\bbl@inikv@##1}{}%
2871     {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2872     \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.

```

2873 \def\bbl@extend@ini@aux#1{%
2874     \bbl@startcommands*{#1}{captions}%
2875     % Activate captions/... and modify exports
2876     \bbl@csarg\def{\inikv@captions.licr}##1##2{%
2877         \setlocalecaption{#1}{##1}{##2}}%
2878     \def\bbl@inikv@captions##1##2{%
2879         \bbl@ini@captions@aux{##1}{##2}}%
2880     \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2881     \def\bbl@exportkey##1##2##3{%
2882         \bbl@ifunset{\bbl@kv@##2}{}%
2883         {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2884             \bbl@exp{\global\let\<bbl@##1@language\>\<bbl@kv@##2>}}%
2885         \fi}}%
2886     % As with \bbl@read@ini, but with some changes
2887     \bbl@read@ini@aux
2888     \bbl@ini@exports\tw@
2889     % Update inidata@lang by pretending the ini is read.
2890     \def\bbl@elt##1##2##3{%
2891         \def\bbl@section{##1}%
2892         \bbl@iniline##2=##3\bbl@iniline}%
2893         \csname bbl@inidata@#1\endcsname
2894         \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2895     \StartBabelCommands*{#1}{date}% And from the import stuff
2896     \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2897     \bbl@savetoday
2898     \bbl@savestate
2899     \bbl@endcommands}

```

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

```

2900 \def\bbl@ini@calendar#1{%
2901     \lowercase{\def\bbl@tempa{=##1=}}%
2902     \bbl@replace\bbl@tempa{=date.gregorian}{}%
2903     \bbl@replace\bbl@tempa{=date.}{}%
2904     \in@{.licr=}{#1=}%
2905     \ifin@
2906         \ifcase\bbl@engine
2907             \bbl@replace\bbl@tempa{.licr=}{}%
2908         \else
2909             \let\bbl@tempa\relax
2910         \fi
2911     \fi
2912     \ifx\bbl@tempa\relax\else
2913         \bbl@replace\bbl@tempa{=}{}%
2914         \ifx\bbl@tempa\@empty\else
2915             \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2916         \fi
2917         \bbl@exp{%
2918             \def\<bbl@inikv@#1>####1####2{%
2919                 \\\bbl@inidata####1...\relax{####2}{\bbl@tempa}}}%
2920     \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).

```

2921 \def\bbl@renewinikey#1/#2\@#3{%
2922   \edef\bbl@tempa{\zap@space #1 \@empty}%   section
2923   \edef\bbl@tempb{\zap@space #2 \@empty}%   key
2924   \bbl@trim\toks@{#3}%                       value
2925   \bbl@exp{%
2926     \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2927     \\g@addto@macro\\bbl@inidata{%
2928       \\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.

```

2929 \def\bbl@exportkey#1#2#3{%
2930   \bbl@ifunset{\bbl@kv@#2}%
2931     {\bbl@csarg\gdef{#1@\language}\@#3}%
2932     {\expandafter\ifx\csname \bbl@kv@#2\endcsname\@empty
2933       \bbl@csarg\gdef{#1@\language}\@#3}%
2934     \else
2935       \bbl@exp{\global\let\<\bbl@#1@\language>\<\bbl@kv@#2>}%
2936       \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.

```

2937 \def\bbl@iniwarning#1{%
2938   \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
2939   {\bbl@warning{%
2940     From babel-\bbl@cs{lini@\language}.ini:\\%
2941     \bbl@cs{@kv@identification.warning#1}\\%
2942     Reported }}}
2943 %
2944 \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.

```

2945 \def\bbl@ini@extension#1{%
2946   \def\bbl@tempa{#1}%
2947   \bbl@replace\bbl@tempa{extension.}{}%
2948   \bbl@replace\bbl@tempa{.tag.bcp47}{}%
2949   \bbl@ifunset{\bbl@info@#1}%
2950     {\bbl@csarg\xdef{info@#1}{ext/\bbl@tempa}%
2951     \bbl@exp{%
2952       \\g@addto@macro\\bbl@moreinfo{%
2953         \\bbl@exportkey{ext/\bbl@tempa}{identification.#1}{}}}%
2954     {}}
2955 \let\bbl@moreinfo\@empty
2956 %
2957 \def\bbl@ini@exports#1{%
2958   % Identification always exported
2959   \bbl@iniwarning{}}%
2960   \ifcase\bbl@engine
2961     \bbl@iniwarning{.pdflatex}%
2962   \or
2963     \bbl@iniwarning{.lua\latex}%
2964   \or
2965     \bbl@iniwarning{.xel\latex}%
2966   \fi%
2967   \bbl@exportkey{llevel}{identification.load.level}{}%
2968   \bbl@exportkey{elname}{identification.name.english}{}%
2969   \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%

```

```

2970   {\csname bbl@elname@\languagename\endcsname}}%
2971   \bbl@exportkey{tbcpr}{identification.tag.bcp47}{}%
2972   \bbl@exportkey{lbcpr}{identification.language.tag.bcp47}{}%
2973   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2974   \bbl@exportkey{esname}{identification.script.name}{}%
2975   \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
2976     {\csname bbl@esname@\languagename\endcsname}}%
2977   \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
2978   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2979   \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2980   \bbl@exportkey{vbcpr}{identification.variant.tag.bcp47}{}%
2981   \bbl@moreinfo
2982   % Also maps bcp47 -> languagename
2983   \ifbbl@bcptoname
2984     \bbl@csarg\xdef{bcp@map@\bbl@cl{tbcpr}}{\languagename}%
2985   \fi
2986   % Conditional
2987   \ifnum#1>\z@           % 0 = only info, 1, 2 = basic, (re)new
2988     \bbl@exportkey{calpr}{date.calendar.preferred}{}%
2989     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2990     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2991     \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
2992     \bbl@exportkey{rgtm}{typography.righthyphenmin}{3}%
2993     \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2994     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2995     \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2996     \bbl@exportkey{intsp}{typography.intraspace}{}%
2997     \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2998     \bbl@exportkey{chrng}{characters.ranges}{}%
2999     \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
3000     \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3001     \ifnum#1=\tw@       % only (re)new
3002       \bbl@exportkey{rqtex}{identification.require.babel}{}%
3003       \bbl@tglobal\bbl@savetoday
3004       \bbl@tglobal\bbl@savestate
3005       \bbl@savestrings
3006     \fi
3007   \fi}

```

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

```

3008 \def\bbl@inikv#1#2{%      key=value
3009   \toks@{#2}%             This hides #'s from ini values
3010   \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}

```

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

```

3011 \let\bbl@inikv@identification\bbl@inikv
3012 \let\bbl@inikv@date\bbl@inikv
3013 \let\bbl@inikv@typography\bbl@inikv
3014 \let\bbl@inikv@characters\bbl@inikv
3015 \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’.

```

3016 \def\bbl@inikv@counters#1#2{%
3017   \bbl@ifsamestring{#1}{digits}%
3018   {\bbl@error{The counter name 'digits' is reserved for mapping\\%
3019     decimal digits}%
3020     {Use another name.}}%
3021   {}%
3022   \def\bbl@tempc{#1}%
3023   \bbl@trim@def{\bbl@tempb*}{#2}%
3024   \in@{.1$}{#1$}%
3025   \ifin@

```

```

3026 \bbl@replace\bbl@tempc{.1}{}%
3027 \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}%
3028 \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3029 \fi
3030 \in@{.F.}{#1}%
3031 \ifin@else\in@{.S.}{#1}\fi
3032 \ifin@
3033 \bbl@csarg\protected@xdef{cntr@#1@\language}{\bbl@tempb*}%
3034 \else
3035 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3036 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3037 \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
3038 \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.

```

3039 \ifcase\bbl@engine
3040 \bbl@csarg\def{inikv@captions.licr}#1#2{%
3041 \bbl@ini@captions@aux{#1}{#2}}
3042 \else
3043 \def\bbl@inikv@captions#1#2{%
3044 \bbl@ini@captions@aux{#1}{#2}}
3045 \fi

```

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

```

3046 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3047 \bbl@replace\bbl@tempa{.template}{}%
3048 \def\bbl@toreplace{#1}{}%
3049 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
3050 \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3051 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3052 \bbl@replace\bbl@toreplace{[ ]}{name\endcsname}}%
3053 \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
3054 \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3055 \ifin@
3056 \@nameuse{\bbl@patch\bbl@tempa}%
3057 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3058 \fi
3059 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3060 \ifin@
3061 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3062 \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
3063 \ \bbl@ifunset{\bbl@\bbl@tempa fmt@\ \ \language}%
3064 {[fnum@\bbl@tempa]}%
3065 {\ \ \nameuse{\bbl@\bbl@tempa fmt@\ \ \language}}}%
3066 \fi}
3067 \def\bbl@ini@captions@aux#1#2{%
3068 \bbl@trim\def\bbl@tempa{#1}%
3069 \bbl@xin@{.template}{\bbl@tempa}%
3070 \ifin@
3071 \bbl@ini@captions@template{#2}\language
3072 \else
3073 \bbl@ifblank{#2}%
3074 {\bbl@exp{%
3075 \toks@{\ \bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
3076 {\bbl@trim\toks@{#2}}%
3077 \bbl@exp{%
3078 \ \bbl@add\ \bbl@savestrings{%
3079 \ \SetString\<\bbl@tempa name>{\the\toks@}}}%
3080 \toks@\expandafter{\bbl@captionslist}%
3081 \bbl@exp{\ \in@{\<\bbl@tempa name>}{\the\toks@}}%
3082 \ifin@else
3083 \bbl@exp{%

```

```

3084      \\bbl@add\<bbl@extracaps@\language\>\<bbl@tempa name\>%
3085      \\bbl@tglobal\<bbl@extracaps@\language\>%
3086      \fi
3087      \fi}

```

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

```

3088 \def\bbl@list@the{%
3089   part,chapter,section,subsection,subsubsection,paragraph,%
3090   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3091   table,page,footnote,mpfootnote,mpfn}
3092 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3093   \bbl@ifunset{bbl@map@#1@\language}%
3094     {\@nameuse{#1}}%
3095     {\@nameuse{bbl@map@#1@\language}}}
3096 \def\bbl@inikv@labels#1#2{%
3097   \in@{.map}{#1}%
3098   \ifin@
3099     \ifx\bbl@KVP@labels\@nnil\else
3100       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3101       \ifin@
3102         \def\bbl@tempc{#1}%
3103         \bbl@replace\bbl@tempc{.map}{}%
3104         \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3105         \bbl@exp{%
3106           \gdef\<bbl@map@\bbl@tempc @\language\>%
3107             {\ifin@\<#2>\else\\localecounter{#2}\fi}}%
3108         \bbl@foreach\bbl@list@the{%
3109           \bbl@ifunset{the##1}{}%
3110             {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3111               \bbl@exp{%
3112                 \\bbl@sreplace\<the##1>%
3113                 {\<\bbl@tempc>{##1}}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3114                 \\bbl@sreplace\<the##1>%
3115                 {\<\@empty @\bbl@tempc>\<c@##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3116               \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3117                 \toks@\expandafter\expandafter\expandafter{%
3118                   \csname the##1\endcsname}%
3119                 \expandafter\xdef\csname the##1\endcsname{\the\toks@}}%
3120               \fi}}%
3121         \fi
3122       \fi
3123     %
3124   \else
3125     %
3126     % The following code is still under study. You can test it and make
3127     % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3128     % language dependent.
3129     \in@{enumerate.}{#1}%
3130     \ifin@
3131       \def\bbl@tempa{#1}%
3132       \bbl@replace\bbl@tempa{enumerate.}{}%
3133       \def\bbl@toreplace{#2}%
3134       \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}%
3135       \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3136       \bbl@replace\bbl@toreplace{ ]}{\endcsname}%
3137       \toks@\expandafter{\bbl@toreplace}%
3138       % TODO. Execute only once:
3139       \bbl@exp{%
3140         \\bbl@add\<extras\language\>%
3141         \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3142         \def\<labelenum\romannumeral\bbl@tempa>\<the\toks@>%
3143         \\bbl@tglobal\<extras\language\>%
3144       \fi

```



```
3145 \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.

```
3146 \def\bb1@chapttype{chapter}
3147 \ifx\@makechapterhead\@undefined
3148 \let\bb1@patchchapter\relax
3149 \else\ifx\thechapter\@undefined
3150 \let\bb1@patchchapter\relax
3151 \else\ifx\ps@headings\@undefined
3152 \let\bb1@patchchapter\relax
3153 \else
3154 \def\bb1@patchchapter{%
3155 \global\let\bb1@patchchapter\relax
3156 \gdef\bb1@chfmt{%
3157 \bb1@ifunset{\bb1@bb1@chapttype fmt@\language name}%
3158 {\@chapapp\space\thechapter}
3159 {\@nameuse{\bb1@bb1@chapttype fmt@\language name}}}%
3160 \bb1@add\appendix{\def\bb1@chapttype{appendix}}% Not harmful, I hope
3161 \bb1@sreplace\ps@headings{\@chapapp\ \thechapter}{\bb1@chfmt}%
3162 \bb1@sreplace\chaptermark{\@chapapp\ \thechapter}{\bb1@chfmt}%
3163 \bb1@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bb1@chfmt}%
3164 \bb1@tglobal\appendix
3165 \bb1@tglobal\ps@headings
3166 \bb1@tglobal\chaptermark
3167 \bb1@tglobal\@makechapterhead}
3168 \let\bb1@patchappendix\bb1@patchchapter
3169 \fi\fi\fi
3170 \ifx\@part\@undefined
3171 \let\bb1@patchpart\relax
3172 \else
3173 \def\bb1@patchpart{%
3174 \global\let\bb1@patchpart\relax
3175 \gdef\bb1@partformat{%
3176 \bb1@ifunset{\bb1@partfmt@\language name}%
3177 {\partname\nobreakspace\thepart}
3178 {\@nameuse{\bb1@partfmt@\language name}}}%
3179 \bb1@sreplace\@part{\partname\nobreakspace\thepart}{\bb1@partformat}%
3180 \bb1@tglobal\@part}
3181 \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

```
3182 \let\bb1@calendar\@empty
3183 \DeclareRobustCommand\localedate[1][\bb1@localedate{#1}]
3184 \def\bb1@localedate#1#2#3#4{%
3185 \begingroup
3186 \edef\bb1@they{#2}%
3187 \edef\bb1@them{#3}%
3188 \edef\bb1@thed{#4}%
3189 \edef\bb1@tempe{%
3190 \bb1@ifunset{\bb1@calpr@\language name}{\bb1@cl{calpr}}{,%
3191 #1}%
3192 \bb1@replace\bb1@tempe{ }{}%
3193 \bb1@replace\bb1@tempe{CONVERT}{convert=}% Hackish
3194 \bb1@replace\bb1@tempe{convert}{convert=}%
3195 \let\bb1@ld@calendar\@empty
3196 \let\bb1@ld@variant\@empty
3197 \let\bb1@ld@convert\relax
3198 \def\bb1@tempb##1=##2\@@{\@namedef{\bb1@ld@##1}{##2}}%
3199 \bb1@foreach\bb1@tempe{\bb1@tempb##1\@@}%
3200 \bb1@replace\bb1@ld@calendar{gregorian}{}%
```

```

3201 \ifx\bbld@calendar\@empty\else
3202 \ifx\bbld@convert\relax\else
3203 \babelcalendar[\bbld@they-\bbld@them-\bbld@thed]%
3204 {\bbld@calendar}\bbld@they\bbld@them\bbld@thed
3205 \fi
3206 \fi
3207 \@nameuse{bbld@precalendar}% Remove, eg, +, -civil (-ca-islamic)
3208 \edef\bbld@calendar{% Used in \month..., too
3209 \bbld@calendar
3210 \ifx\bbld@variant\@empty\else
3211 .\bbld@variant
3212 \fi}%
3213 \bbld@cased
3214 {\@nameuse{bbld@date@\language name @\bbld@calendar}%
3215 \bbld@they\bbld@them\bbld@thed}%
3216 \endgroup}
3217 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3218 \def\bbld@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3219 \bbld@trim@def\bbld@tempa{#1.#2}%
3220 \bbld@ifsamestring{\bbld@tempa}{months.wide}% to savedate
3221 {\bbld@trim@def\bbld@tempa{#3}%
3222 \bbld@trim\toks@{#5}%
3223 \@temptokena\expandafter{\bbld@savedate}%
3224 \bbld@exp{% Reverse order - in ini last wins
3225 \def\\bbld@savedate{%
3226 \\SetString\<month\romannumeral\bbld@tempa#6name>{\the\toks@}%
3227 \the\@temptokena}}}%
3228 {\bbld@ifsamestring{\bbld@tempa}{date.long}% defined now
3229 {\lowercase{\def\bbld@tempb{#6}}}%
3230 \bbld@trim@def\bbld@toreplace{#5}%
3231 \bbld@TG@date
3232 \global\bbld@csarg\let{date@\language name @\bbld@tempb}\bbld@toreplace
3233 \ifx\bbld@savetoday\@empty
3234 \bbld@exp{% TODO. Move to a better place.
3235 \\AfterBabelCommands{%
3236 \def\<\language name date>{\\protect\<\language name date >}%
3237 \\newcommand\<\language name date >[4][]{%
3238 \\bbld@usedategroupttrue
3239 \<bbld@ensure@\language name>{%
3240 \\localedate[####1]{####2}{####3}{####4}}}%
3241 \def\\bbld@savetoday{%
3242 \\SetString\\today{%
3243 \<\language name date>[convert]%
3244 {\the\year}{\the\month}{\the\day}}}%
3245 \fi}%
3246 {}}}}

```

**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 \bbld@replace \toks@ contains the resulting string, which is used by \bbld@replace@finish@iii (this implicit behavior doesn’t seem a good idea, but it’s efficient).

```

3247 \let\bbld@calendar\@empty
3248 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3249 \@nameuse{bbld@ca#2}#1\@}
3250 \newcommand\BabelDateSpace{\nobreakspace}
3251 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3252 \newcommand\BabelDated[1]{\number#1}
3253 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3254 \newcommand\BabelDateM[1]{\number#1}
3255 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3256 \newcommand\BabelDateMMM[1]{%
3257 \csname month\romannumeral#1\bbld@calendar name\endcsname}%

```

```

3258 \newcommand\BabelDatey[1]{\number#1}%
3259 \newcommand\BabelDateyy[1]{%
3260   \ifnum#1<10 0\number#1 %
3261   \else\ifnum#1<100 \number#1 %
3262   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3263   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3264   \else
3265     \bbl@error
3266     {Currently two-digit years are restricted to the\
3267      range 0-9999.}%
3268     {There is little you can do. Sorry.}%
3269   \fi\fi\fi\fi}}
3270 \newcommand\BabelDateyyyy[1]{\number#1} % TODO - add leading 0
3271 \def\bbl@replace@finish@iii#1{%
3272   \bbl@exp{\def\#1###1###2###3{\the\toks@}}
3273 \def\bbl@TG@@date{%
3274   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3275   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3276   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{###3}}%
3277   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{###3}}%
3278   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{###2}}%
3279   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{###2}}%
3280   \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMM{###2}}%
3281   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{###1}}%
3282   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{###1}}%
3283   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{###1}}%
3284   \bbl@replace\bbl@toreplace{[y]}{\bbl@datecctr[###1|]}%
3285   \bbl@replace\bbl@toreplace{[m]}{\bbl@datecctr[###2|]}%
3286   \bbl@replace\bbl@toreplace{[d]}{\bbl@datecctr[###3|]}%
3287   \bbl@replace@finish@iii\bbl@toreplace}
3288 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3289 \def\bbl@xdatecctr[#1|#2]{\localenumeral{#2}{#1}}

```

### Transforms.

```

3290 \let\bbl@release@transforms\@empty
3291 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3292 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3293 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3294   #1[#2]{#3}{#4}{#5}}
3295 \begingroup % A hack. TODO. Don't require an specific order
3296   \catcode`\%=12
3297   \catcode`\&=14
3298   \gdef\bbl@transforms#1#2#3{%&
3299     \directlua{
3300       local str = [==[#2]==]
3301       str = str:gsub('%.%d+%.%d+$', '')
3302       token.set_macro('babeltempa', str)
3303     }&
3304     \def\babeltempc{}&
3305     \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&
3306     \ifin@ \else
3307       \bbl@xin@{: \babeltempa,}{,\bbl@KVP@transforms,}&
3308     \fi
3309     \ifin@
3310       \bbl@foreach\bbl@KVP@transforms{%&
3311         \bbl@xin@{: \babeltempa,}{,##1,}&
3312         \ifin@ & font:font:transform syntax
3313         \directlua{
3314           local t = {}
3315           for m in string.gmatch('##1'..' ':'(.-):') do
3316             table.insert(t, m)
3317           end
3318           table.remove(t)

```

```

3319         token.set_macro('babeltempc', ',font=' .. table.concat(t, ' '))
3320     }&%
3321     \fi}&%
3322     \in@{.0$}{#2$}&%
3323     \ifin@
3324         \directlua{&% (\attribute) syntax
3325             local str = string.match([[\\bbl@KVP@transforms]],
3326                 '%([^(%-)]%)[^%)]-\\babeltempa')
3327             if str == nil then
3328                 token.set_macro('babeltempb', '')
3329             else
3330                 token.set_macro('babeltempb', ',attribute=' .. str)
3331             end
3332         }&%
3333         \toks@{#3}&%
3334         \bbl@exp{&%
3335             \\g@addto@macro\\bbl@release@transforms{&%
3336                 \relax &% Closes previous \bbl@transforms@aux
3337                 \\bbl@transforms@aux
3338                 \\#1{label=\\babeltempa\\babeltempb\\babeltempc}&%
3339                 {\\languagenamename}{\\the\\toks@}}}&%
3340         \else
3341             \g@addto@macro\\bbl@release@transforms{, {#3}}&%
3342         \fi
3343     \fi}
3344 \endgroup

```

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

```

3345 \def\bbl@provide@lsys#1{%
3346     \bbl@ifunset{bbl@lname@#1}%
3347     {\bbl@load@info{#1}}%
3348     {}%
3349     \bbl@csarg\let{lsys@#1}\@empty
3350     \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}}%
3351     \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}}%
3352     \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3353     \bbl@ifunset{bbl@lname@#1}{%
3354         {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3355     \ifcase\bbl@engine\or\or
3356         \bbl@ifunset{bbl@prehc@#1}{%
3357             {\bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3358             {}%
3359             {\ifx\bbl@xenoxyph\undefined
3360                 \global\let\bbl@xenoxyph\bbl@xenoxyph@d
3361                 \ifx\AtBeginDocument\notprerr
3362                     \expandafter\@secondoftwo % to execute right now
3363                 \fi
3364                 \AtBeginDocument{%
3365                     \bbl@patchfont{\bbl@xenoxyph}%
3366                     \expandafter\selectlanguage\expandafter{\languagenamename}}%
3367             \fi}}%
3368     \fi
3369     \bbl@csarg\bbl@tglobal{lsys@#1}}
3370 \def\bbl@xenoxyph@d{%
3371     \bbl@ifset{bbl@prehc@languagenamename}%
3372     {\ifnum\hyphenchar\font=\defaultshyphenchar
3373         \iffontchar\font\bbl@cl{prehc}\relax
3374         \hyphenchar\font\bbl@cl{prehc}\relax
3375     \else\iffontchar\font"200B
3376         \hyphenchar\font"200B
3377     \else
3378         \bbl@warning

```

```

3379         {Neither 0 nor ZERO WIDTH SPACE are available\\%
3380         in the current font, and therefore the hyphen\\%
3381         will be printed. Try changing the fontspec's\\%
3382         'HyphenChar' to another value, but be aware\\%
3383         this setting is not safe (see the manual).\\%
3384         Reported}%
3385         \hyphenchar\font\defaultthyphenchar
3386         \fi\fi
3387         \fi}%
3388         {\hyphenchar\font\defaultthyphenchar}}
3389     % \fi}

```

```

3390 \def\bbl@load@info#1{%
3391   \def\BabelBeforeIni##1##2{%
3392     \begingroup
3393       \bbl@read@ini{##1}0%
3394       \endinput           % babel- .tex may contain onlypreamble's
3395       \endgroup}%         boxed, to avoid extra spaces:
3396   {\bbl@input@texini{#1}}}
```

```

3397 \def\bbl@setdigits#1#2#3#4#5{%
3398   \bbl@exp{%
3399     \def\<\language name digits>####1{%       ie, \lang digits
3400       \<\bbl@digits@\language name>####1\\\@nil}%
3401       \let\<\bbl@cnt@digits@\language name>\<\language name digits>%
3402       \def\<\language name counter>####1{%       ie, \lang counter
3403         \\\expandafter\<\bbl@counter@\language name>%
3404         \\\csname c@####1\endcsname}%
3405       \def\<\bbl@counter@\language name>####1{% ie, \bbl@counter@lang
3406         \\\expandafter\<\bbl@digits@\language name>%
3407         \\\number####1\\\@nil}%}
3408 \def\bbl@tempa##1##2##3##4##5{%
3409   \bbl@exp{%    Wow, quite a lot of hashes! :- (
3410     \def\<\bbl@digits@\language name>#####1{%
3411       \\\ifx#####1\\\@nil                % ie, \bbl@digits@lang
3412       \\\else
3413         \\\ifx0#####1#1%
3414         \\\else\\\ifx1#####1#2%
3415         \\\else\\\ifx2#####1#3%
3416         \\\else\\\ifx3#####1#4%
3417         \\\else\\\ifx4#####1#5%
3418         \\\else\\\ifx5#####1##1%
3419         \\\else\\\ifx6#####1##2%
3420         \\\else\\\ifx7#####1##3%
3421         \\\else\\\ifx8#####1##4%
3422         \\\else\\\ifx9#####1##5%
3423         \\\else#####1%
3424         \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
3425         \\\expandafter\<\bbl@digits@\language name>%
3426         \\\fi}}}%
3427 \bbl@tempa}

```

```

3428 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}%
3429   \ifx\#\#1%           % \ before, in case #1 is multiletter
3430     \bbl@exp{%
3431       \def\#\bbl@tempa####1{%

```

```

3432 \<ifcase>####1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
3433 \else
3434 \toks@\expandafter{\the\toks@\or #1}%
3435 \expandafter\bb1@buildifcase
3436 \fi}

```

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

```

3437 \newcommand\localenumeral[2]{\bb1@cs{cntr@#1@\language}\{#2}}
3438 \def\bb1@localecntr#1#2{\localenumeral{#2}{#1}}
3439 \newcommand\localecounter[2]{%
3440 \expandafter\bb1@localecntr
3441 \expandafter{\number\csname c@#2\endcsname}\{#1}}
3442 \def\bb1@alphnumeral#1#2{%
3443 \expandafter\bb1@alphnumeral@i\number#2 76543210\@@{#1}}
3444 \def\bb1@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3445 \ifcase\@car#8\@nil\or % Currently <10000, but prepared for bigger
3446 \bb1@alphnumeral@ii{#9}000000#1\or
3447 \bb1@alphnumeral@ii{#9}00000#1#2\or
3448 \bb1@alphnumeral@ii{#9}0000#1#2#3\or
3449 \bb1@alphnumeral@ii{#9}000#1#2#3#4\else
3450 \bb1@alphnum@invalid{>9999}%
3451 \fi}
3452 \def\bb1@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3453 \bb1@ifunset{bb1@cntr@#1.F.\number#5#6#7#8@\language}%
3454 {\bb1@cs{cntr@#1.4@\language}\{#5}
3455 \bb1@cs{cntr@#1.3@\language}\{#6}
3456 \bb1@cs{cntr@#1.2@\language}\{#7}
3457 \bb1@cs{cntr@#1.1@\language}\{#8}
3458 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3459 \bb1@ifunset{bb1@cntr@#1.S.321@\language}\{#}
3460 {\bb1@cs{cntr@#1.S.321@\language}\{#}
3461 \fi}%
3462 {\bb1@cs{cntr@#1.F.\number#5#6#7#8@\language}\{#}
3463 \def\bb1@alphnum@invalid#1{%
3464 \bb1@error{Alphabetic numeral too large (#1)}%
3465 {Currently this is the limit.}}

```

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

```

3466 \def\bb1@localeinfo#1#2{%
3467 \bb1@ifunset{bb1@info@#2}\{#1}%
3468 {\bb1@ifunset{bb1@csname bb1@info@#2\endcsname @\language}\{#1}%
3469 {\bb1@cs{csname bb1@info@#2\endcsname @\language}\{#1}}
3470 \newcommand\localeinfo[1]{%
3471 \ifx*#1\@empty % TODO. A bit hackish to make it expandable.
3472 \bb1@afterelse\bb1@localeinfo}%
3473 \else
3474 \bb1@localeinfo
3475 {\bb1@error{I've found no info for the current locale.\\%
3476 The corresponding ini file has not been loaded\\%
3477 Perhaps it doesn't exist}%
3478 {See the manual for details.}}%
3479 {#1}%
3480 \fi}
3481 % \@namedef{bb1@info@name.locale}\{lname}
3482 \@namedef{bb1@info@tag.ini}\{lini}
3483 \@namedef{bb1@info@name.english}\{elname}
3484 \@namedef{bb1@info@name.opentype}\{lname}
3485 \@namedef{bb1@info@tag.bcp47}\{tbc47}
3486 \@namedef{bb1@info@language.tag.bcp47}\{lbc47}

```

```

3487 \@namedef{bbl@info@tag.opentype}{lotf}
3488 \@namedef{bbl@info@script.name}{esname}
3489 \@namedef{bbl@info@script.name.opentype}{sname}
3490 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3491 \@namedef{bbl@info@script.tag.opentype}{sotf}
3492 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3493 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3494 % Extensions are dealt with in a special way
3495 % Now, an internal \LaTeX{} macro:
3496 \providecommand\BCPdata[1]{\localeinfo*{#1.tag.bcp47}}

```

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

```

3497 <(*More package options)> \equiv
3498 \DeclareOption{ensureinfo=off}{}
3499 <(/More package options)>
3500 %
3501 \let\bbl@ensureinfo\@gobble
3502 \newcommand\BabelEnsureInfo{%
3503   \ifx\InputIfFileExists\undefined\else
3504     \def\bbl@ensureinfo##1{%
3505       \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3506   \fi
3507   \bbl@foreach\bbl@loaded{%
3508     \def\language{##1}%
3509     \bbl@ensureinfo{##1}}}%
3510 \@ifpackagewith{babel}{ensureinfo=off}{}%
3511 {\AtEndOfPackage{% Test for plain.
3512   \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.

```

3513 \newcommand\getlocaleproperty{%
3514   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3515 \def\bbl@getproperty@s#1#2#3{%
3516   \let#1\relax
3517   \def\bbl@elt##1##2##3{%
3518     \bbl@ifsamestring{##1/##2}{#3}%
3519     {\providecommand#1{##3}%
3520     \def\bbl@elt####1####2####3{}}}%
3521   {}}%
3522   \bbl@cs{inidata@#2}}%
3523 \def\bbl@getproperty@x#1#2#3{%
3524   \bbl@getproperty@s{#1}{#2}{#3}%
3525   \ifx#1\relax
3526     \bbl@error
3527     {Unknown key for locale '#2':\%
3528     #3\}%
3529     \string#1 will be set to \relax}%
3530     {Perhaps you misspelled it.}%
3531   \fi}
3532 \let\bbl@ini@loaded\@empty
3533 \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.

```

3534 \newcommand\babeladjust[1]{% TODO. Error handling.
3535   \bbl@forkv{#1}{%
3536     \bbl@ifunset{bbl@ADJ@##1@##2}%
3537     {\bbl@cs{ADJ@##1}{##2}}%
3538     {\bbl@cs{ADJ@##1@##2}}}

```

```

3539 %
3540 \def\bbl@adjust@lua#1#2{%
3541   \ifvmode
3542     \ifnum\currentgrouplevel=\z@
3543       \directlua{ Babel.#2 }%
3544       \expandafter\expandafter\expandafter\@gobble
3545       \fi
3546   \fi
3547   {\bbl@error % The error is gobbled if everything went ok.
3548     {Currently, #1 related features can be adjusted only\\%
3549       in the main vertical list.}%
3550     {Maybe things change in the future, but this is what it is.}}}
3551 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3552   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3553 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3554   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3555 \@namedef{bbl@ADJ@bidi.text@on}{%
3556   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3557 \@namedef{bbl@ADJ@bidi.text@off}{%
3558   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3559 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3560   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3561 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3562   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3563 %
3564 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3565   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3566 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3567   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3568 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3569   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3570 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3571   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3572 \@namedef{bbl@ADJ@justify.arabic@on}{%
3573   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3574 \@namedef{bbl@ADJ@justify.arabic@off}{%
3575   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3576 %
3577 \def\bbl@adjust@layout#1{%
3578   \ifvmode
3579     #1%
3580     \expandafter\@gobble
3581   \fi
3582   {\bbl@error % The error is gobbled if everything went ok.
3583     {Currently, layout related features can be adjusted only\\%
3584       in vertical mode.}%
3585     {Maybe things change in the future, but this is what it is.}}}
3586 \@namedef{bbl@ADJ@layout.tabular@on}{%
3587   \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}}
3588 \@namedef{bbl@ADJ@layout.tabular@off}{%
3589   \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}}
3590 \@namedef{bbl@ADJ@layout.lists@on}{%
3591   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3592 \@namedef{bbl@ADJ@layout.lists@off}{%
3593   \bbl@adjust@layout{\let\list\bbl@OL@list}}
3594 %
3595 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3596   \bbl@bcpallowedtrue}
3597 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3598   \bbl@bcpallowedfalse}
3599 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3600   \def\bbl@bcp@prefix{#1}}
3601 \def\bbl@bcp@prefix{bcp47-}

```



```

3602 \@namedef{bbl@ADJ@autoload.options}#1{%
3603   \def\bbl@autoload@options{#1}}
3604 \let\bbl@autoload@bcptoptions\@empty
3605 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3606   \def\bbl@autoload@bcptoptions{#1}}
3607 \newif\ifbbl@bcptname
3608 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3609   \bbl@bcptnametrue
3610   \BabelEnsureInfo}
3611 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3612   \bbl@bcptnamefalse}
3613 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3614   \directlua{ Babel.ignore_pre_char = function(node)
3615     return (node.lang == \the\csname l@nohyphenation\endcsname)
3616   end }}
3617 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3618   \directlua{ Babel.ignore_pre_char = function(node)
3619     return false
3620   end }}
3621 \@namedef{bbl@ADJ@select.write@shift}{%
3622   \let\bbl@restorelastskip\relax
3623   \def\bbl@savelastskip{%
3624     \let\bbl@restorelastskip\relax
3625     \ifvmode
3626       \ifdim\lastskip=\z@
3627         \let\bbl@restorelastskip\nobreak
3628       \else
3629         \bbl@exp{%
3630           \def\\bbl@restorelastskip{%
3631             \skip@=\the\lastskip
3632             \\nobreak \vskip-\skip@ \vskip\skip@}}%
3633       \fi
3634     \fi}}
3635 \@namedef{bbl@ADJ@select.write@keep}{%
3636   \let\bbl@restorelastskip\relax
3637   \let\bbl@savelastskip\relax}
3638 \@namedef{bbl@ADJ@select.write@omit}{%
3639   \AddBabelHook{babel-select}{beforestart}{%
3640     \expandafter\babel@aux\expandafter{\bbl@main@language}}}%
3641   \let\bbl@restorelastskip\relax
3642   \def\bbl@savelastskip##1\bbl@restorelastskip{}}
3643 \@namedef{bbl@ADJ@select.encoding@off}{%
3644   \let\bbl@encoding@select@off\@empty}

```

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

```

3645 \ifx\directlua\@undefined\else
3646   \ifx\bbl@luapatterns\@undefined
3647     \input luababel.def
3648   \fi
3649 \fi

```

Continue with  $\LaTeX$ .

```

3650 </package | core>
3651 <*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.

```
3652 <<(*More package options)>> ≡
3653 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3654 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3655 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3656 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3657 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3658 <</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.

```
3659 \bbl@trace{Cross referencing macros}
3660 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3661   \def\@newl@bel#1#2#3{%
3662     {\@safe@activestrue
3663       \bbl@ifunset{#1@#2}%
3664       \relax
3665       {\gdef\@multiplelabels{%
3666         \@latex@warning@no@line{There were multiply-defined labels}}%
3667         \@latex@warning@no@line{Label `#2' multiply defined}}%
3668       \global\@namedef{#1@#2}{#3}}}
```

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

```
3669 \CheckCommand*\@testdef[3]{%
3670   \def\reserved@a{#3}%
3671   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3672   \else
3673     \@tempswatrue
3674   \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.

```
3675 \def\@testdef#1#2#3{% TODO. With @samestring?
3676   \@safe@activestrue
3677   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3678   \def\bbl@tempb{#3}%
3679   \@safe@activesfalse
3680   \ifx\bbl@tempa\relax
3681   \else
3682     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3683   \fi
3684   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3685   \ifx\bbl@tempa\bbl@tempb
3686   \else
3687     \@tempswatrue
3688   \fi}
3689 \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.

```
3690 \bbl@xin@{R}\bbl@opt@safe
3691 \ifin@
3692   \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3693   \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%

```

```

3694     {\expandafter\strip@prefix\meaning\ref}%
3695 \ifin@
3696 \bbl@redefine\@kernel@ref#1{%
3697     \@safe@activetrue\org@@kernel@ref{#1}\@safe@activfalse}
3698 \bbl@redefine\@kernel@pageref#1{%
3699     \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activfalse}
3700 \bbl@redefine\@kernel@sref#1{%
3701     \@safe@activetrue\org@@kernel@sref{#1}\@safe@activfalse}
3702 \bbl@redefine\@kernel@spageref#1{%
3703     \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activfalse}
3704 \else
3705 \bbl@redefineroobust\ref#1{%
3706     \@safe@activetrue\org@ref{#1}\@safe@activfalse}
3707 \bbl@redefineroobust\pageref#1{%
3708     \@safe@activetrue\org@pageref{#1}\@safe@activfalse}
3709 \fi
3710 \else
3711 \let\org@ref\ref
3712 \let\org@pageref\pageref
3713 \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.

```

3714 \bbl@xin@{B}\bbl@opt@safe
3715 \ifin@
3716 \bbl@redefine\@citex[#1]#2{%
3717     \@safe@activetrue\edef\@tempa{#2}\@safe@activfalse
3718     \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.

```

3719 \AtBeginDocument{%
3720     \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.)

```

3721 \def\@citex[#1][#2]#3{%
3722     \@safe@activetrue\edef\@tempa{#3}\@safe@activfalse
3723     \org@@citex[#1][#2]{\@tempa}}%
3724 }{}

```

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

```

3725 \AtBeginDocument{%
3726     \ifpackageloaded{cite}{%
3727         \def\@citex[#1]#2{%
3728             \@safe@activetrue\org@@citex[#1][#2]\@safe@activfalse}%
3729         }{}

```

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

```

3730 \bbl@redefine\nocite#1{%
3731     \@safe@activetrue\org@nocite{#1}\@safe@activfalse}

```

`\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 \bibtex is needed we define \bibtex in such a way that it redefines itself with the proper definition. We call \bbl@cite@choice to select the proper definition for \bibtex. This new definition is then activated.

```
3732 \bbl@redefine\bibtex{%
3733 \bbl@cite@choice
3734 \bibtex}
```

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

```
3735 \def\bbl@bibtex#1#2{%
3736 \org@bibtex{#1}{\@safe@activesfalse#2}}
```

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

```
3737 \def\bbl@cite@choice{%
3738 \global\let\bibtex\bbl@bibtex
3739 \@ifpackageloaded{natbib}{\global\let\bibtex\org@bibtex}}%
3740 \@ifpackageloaded{cite}{\global\let\bibtex\org@bibtex}}%
3741 \global\let\bbl@cite@choice\relax}
```

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

```
3742 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3743 \bbl@redefine\@bibitem#1{%
3744 \@safe@activestrue\org@bibitem{#1}\@safe@activesfalse}
3745 \else
3746 \let\org@nocite\nocite
3747 \let\org@citex\citex
3748 \let\org@bibtex\bibtex
3749 \let\org@@bibitem\@bibitem
3750 \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.

```
3751 \bbl@trace{Marks}
3752 \IfBabelLayout{sectioning}
3753 {\ifx\bbl@opt@headfoot\@nnil
3754 \g@addto@macro\resetactivechars{%
3755 \set@typeset@protect
3756 \expandafter\select@language@\expandafter{\bbl@main@language}%
3757 \let\protect\noexpand
3758 \ifcase\bbl@bidimode\else % Only with bidi. See also above
3759 \edef\thepage{%
3760 \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3761 \fi}%
3762 \fi}
3763 {\ifbbl@single\else
3764 \bbl@ifunset{markright }{\bbl@redefine\bbl@redefineroobust
3765 \markright#1{%
3766 \bbl@ifblank{#1}%
3767 {\org@markright{}}}%
3768 {\toks@{#1}%
3769 \bbl@exp{%
3770 \org@markright{\protect\foreignlanguage{\language}%
3771 {\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.)

```

3772 \ifx\@mkboth\markboth
3773 \def\bbl@tempc{\let\@mkboth\markboth}%
3774 \else
3775 \def\bbl@tempc{}%
3776 \fi
3777 \bbl@ifunset{markboth}{\bbl@redefine\bbl@redefineroobust
3778 \markboth#1#2{%
3779 \protected@edef\bbl@tempb##1{%
3780 \protect\foreignlanguage
3781 {\language\name}\protect\bbl@restore@actives##1}}%
3782 \bbl@ifblank{#1}%
3783 {\toks@{}}%
3784 {\toks@\expandafter{\bbl@tempb{#1}}}%
3785 \bbl@ifblank{#2}%
3786 {\@temptokena{}}%
3787 {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3788 \bbl@exp{\org@markboth{\the\toks@}{\the\@temptokena}}}%
3789 \bbl@tempc
3790 \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.

```

3791 \bbl@trace{Preventing clashes with other packages}
3792 \ifx\org@ref\undefined\else
3793 \bbl@xin@{R}\bbl@opt@safe
3794 \ifin@
3795 \AtBeginDocument{%
3796 \@ifpackageloaded{ifthen}{%
3797 \bbl@redefine@long\ifthenelse#1#2#3{%
3798 \let\bbl@temp@pref\pageref
3799 \let\pageref\org@pageref
3800 \let\bbl@temp@ref\ref
3801 \let\ref\org@ref
3802 \@safe@activestrue
3803 \org@ifthenelse{#1}%
3804 {\let\pageref\bbl@temp@pref
3805 \let\ref\bbl@temp@ref
3806 \@safe@activesfalse
3807 #2}%

```

```

3808      {\let\pageref\bb1@temp@pref
3809      \let\ref\bb1@temp@ref
3810      \@safe@activesfalse
3811      #3}%
3812    }%
3813  }{}%
3814 }
3815 \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`.

```

3816 \AtBeginDocument{%
3817   \@ifpackageloaded{varioref}{%
3818     \bbl@redefine\@vpageref#1[#2]#3{%
3819       \@safe@activestrue
3820       \org@@vpageref{#1}[#2]#3}%
3821     \@safe@activesfalse}%
3822   \bbl@redefine\vrefpagemum#1#2{%
3823     \@safe@activestrue
3824     \org@vrefpagemum{#1}#2}%
3825   \@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.

```

3826   \expandafter\def\csname Ref\endcsname#1{%
3827     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3828   }{}%
3829 }
3830 \fi

```

### 8.3.3 hhline

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

```

3831 \AtEndOfPackage{%
3832   \AtBeginDocument{%
3833     \@ifpackageloaded{hhline}%
3834     {\expandafter\ifx\csname normal@char\string\endcsname\relax
3835       \else
3836         \makeatletter
3837         \def\@currname{hhline}\input{hhline.sty}\makeatother
3838       \fi}%
3839     {}}}

```

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

```

3840 \def\substitutefontfamily#1#2#3{%
3841   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3842   \immediate\write15{%
3843     \string\ProvidesFile{#1#2.fd}%
3844     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}]
3845     \space generated font description file]^^J

```

```

3846 \string\DeclareFontFamily{#1}{#2}{}}^^J
3847 \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}}^^J
3848 \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}}^^J
3849 \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}}^^J
3850 \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}}^^J
3851 \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}}^^J
3852 \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}}^^J
3853 \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}}^^J
3854 \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}}^^J
3855 }%
3856 \closeout15
3857 }
3858 \@onlypreamble\substitutefontfamily

```

## 8.4 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of  $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  and  $\mathrm{L}_{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$  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`

```

3859 \bbl@trace{Encoding and fonts}
3860 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3861 \newcommand\BabelNonText{TS1,T3,TS3}
3862 \let\org@TeX\TeX
3863 \let\org@LaTeX\LaTeX
3864 \let\ensureascii\@firstofone
3865 \AtBeginDocument{%
3866   \def\@elt#1{,#1,}%
3867   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3868   \let\@elt\relax
3869   \let\bbl@tempb\@empty
3870   \def\bbl@tempc{OT1}%
3871   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3872     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3873   \bbl@foreach\bbl@tempa{%
3874     \bbl@xin@{#1}{\BabelNonASCII}%
3875     \ifin@
3876       \def\bbl@tempb{#1}% Store last non-ascii
3877     \else\bbl@xin@{#1}{\BabelNonText}% Pass
3878     \ifin@\else
3879       \def\bbl@tempc{#1}% Store last ascii
3880     \fi
3881   \fi}%
3882   \ifx\bbl@tempb\@empty\else
3883     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3884     \ifin@\else
3885       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3886     \fi
3887   \edef\ensureascii#1{%
3888     {\noexpand\fontencoding{\bbl@tempc}\noexpand\selectfont#1}}%
3889   \DeclareTextCommandDefault{\TeX}{\ensureascii\org@TeX}%
3890   \DeclareTextCommandDefault{\LaTeX}{\ensureascii\org@LaTeX}%
3891   \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.

```

3892 \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.

```

3893 \AtBeginDocument{%
3894   \@ifpackageloaded{fontspec}%
3895     {\xdef\latinencoding{%
3896       \ifx\UTFencname\undefined
3897         EU\ifcase\bbl@engine\or2\or1\fi
3898       \else
3899         \UTFencname
3900       \fi}}%
3901   {\gdef\latinencoding{OT1}%
3902     \ifx\cf@encoding\bbl@t@one
3903       \xdef\latinencoding{\bbl@t@one}%
3904     \else
3905       \def\@elt#1{,#1,}%
3906       \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3907       \let\@elt\relax
3908       \bbl@xin@{,T1,}\bbl@tempa
3909       \ifin@
3910         \xdef\latinencoding{\bbl@t@one}%
3911       \fi
3912     \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.

```

3913 \DeclareRobustCommand{\latintext}{%
3914   \fontencoding{\latinencoding}\selectfont
3915   \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.

```

3916 \ifx\@undefined\DeclareTextFontCommand
3917   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3918 \else
3919   \DeclareTextFontCommand{\textlatin}{\latintext}
3920 \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.

```

3921 \def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}

```

## 8.5 Basic bidi support

**Work in progress.** This code is currently placed here for practical reasons. It will be moved to the correct place soon, I hope.

It is loosely based on `rlbabel.def`, but most of it has been developed from scratch. This babel module (by Johannes Braams and Boris Lavva) has served the purpose of typesetting R documents for two decades, and despite its flaws I think it is still a good starting point (some parts have been copied here almost verbatim), partly thanks to its simplicity. I’ve also looked at `ARABI` (by Youssef Jabri), which is compatible with babel.

There are two ways of modifying macros to make them “bidi”, namely, by patching the internal low-level macros (which is what I have done with lists, columns, counters, tocs, much like `rlbabel` did), and by introducing a “middle layer” just below the user interface (sectioning, footnotes).

- `pdftex` provides a minimal support for bidi text, and it must be done by hand. Vertical typesetting is not possible.
- `xetex` is somewhat better, thanks to its font engine (even if not always reliable) and a few additional tools. However, very little is done at the paragraph level. Another challenging problem is text direction does not honour  $\text{\TeX}$  grouping.



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

```

3922 \bbl@trace{Loading basic (internal) bidi support}
3923 \ifodd\bbl@engine
3924 \else % TODO. Move to txtbabel
3925   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3926     \bbl@error
3927       {The bidi method 'basic' is available only in\\%
3928         luatex. I'll continue with 'bidi=default', so\\%
3929         expect wrong results}%
3930       {See the manual for further details.}%
3931   \let\bbl@beforeforeign\leavevmode
3932   \AtEndOfPackage{%
3933     \EnableBabelHook{babel-bidi}%
3934     \bbl@xebidipar}
3935 \fi\fi
3936 \def\bbl@loadxebidi#1{%
3937   \ifx\RTLfootnotetext\@undefined
3938     \AtEndOfPackage{%
3939       \EnableBabelHook{babel-bidi}%
3940       \bbl@loadfontspec % bidi needs fontspec
3941       \usepackage#1{bidi}}%
3942   \fi}
3943 \ifnum\bbl@bidimode>200
3944   \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3945     \bbl@tentative{bidi=bidi}
3946     \bbl@loadxebidi{}
3947   \or
3948     \bbl@loadxebidi{[rldocument]}
3949   \or
3950     \bbl@loadxebidi{}
3951   \fi
3952 \fi
3953 \fi
3954 % TODO? Separate:
3955 \ifnum\bbl@bidimode=\@ne
3956   \let\bbl@beforeforeign\leavevmode
3957   \ifodd\bbl@engine
3958     \newattribute\bbl@attr@dir
3959     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
3960     \bbl@exp{\output{\bodydir\pagedir\the\output}}
3961   \fi
3962   \AtEndOfPackage{%
3963     \EnableBabelHook{babel-bidi}%
3964     \ifodd\bbl@engine\else
3965       \bbl@xebidipar
3966     \fi}
3967 \fi

```

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

```

3968 \bbl@trace{Macros to switch the text direction}
3969 \def\bbl@alscripts{Arabic,Syriac,Thaana,}
3970 \def\bbl@rscripts{% TODO. Base on codes ??
3971   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
3972   Old Hungarian,Lydian,Mandaean,Manichaean,%
3973   Meroitic Cursive,Meroitic,Old North Arabian,%
3974   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
3975   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
3976   Old South Arabian,}%
3977 \def\bbl@provide@dirs#1{%

```

```

3978 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
3979 \ifin@
3980 \global\bbl@csarg\chardef{wdir@#1}\@ne
3981 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
3982 \ifin@
3983 \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
3984 \fi
3985 \else
3986 \global\bbl@csarg\chardef{wdir@#1}\z@
3987 \fi
3988 \ifodd\bbl@engine
3989 \bbl@csarg\ifcase{wdir@#1}%
3990 \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
3991 \or
3992 \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
3993 \or
3994 \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
3995 \fi
3996 \fi}
3997 \def\bbl@switchdir{%
3998 \bbl@ifunset{\bbl@lsys@{\languagename}}{\bbl@provide@lsys@{\languagename}}{}%
3999 \bbl@ifunset{\bbl@wdir@{\languagename}}{\bbl@provide@dirs@{\languagename}}{}%
4000 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}}%
4001 \def\bbl@setdirs#1{% TODO - math
4002 \ifcase\bbl@select@type % TODO - strictly, not the right test
4003 \bbl@bodydir{#1}%
4004 \bbl@pardir{#1}%
4005 \fi
4006 \bbl@textdir{#1}}
4007 % TODO. Only if \bbl@bidimode > 0?:
4008 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4009 \DisableBabelHook{babel-bidi}

```

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

```

4010 \ifodd\bbl@engine % luatex=1
4011 \else % pdftex=0, xetex=2
4012 \newcount\bbl@dirlevel
4013 \chardef\bbl@thetextdir\z@
4014 \chardef\bbl@thepardir\z@
4015 \def\bbl@textdir#1{%
4016 \ifcase#1\relax
4017 \chardef\bbl@thetextdir\z@
4018 \bbl@textdir@i\beginL\endL
4019 \else
4020 \chardef\bbl@thetextdir\@ne
4021 \bbl@textdir@i\beginR\endR
4022 \fi}
4023 \def\bbl@textdir@i#1#2{%
4024 \ifhmode
4025 \ifnum\currentgrouplevel>\z@
4026 \ifnum\currentgrouplevel=\bbl@dirlevel
4027 \bbl@error{Multiple bidi settings inside a group}%
4028 {I'll insert a new group, but expect wrong results.}%
4029 \bgroup\aftergroup#2\aftergroup\egroup
4030 \else
4031 \ifcase\currentgrouptype\or % 0 bottom
4032 \aftergroup#2% 1 simple {}
4033 \or
4034 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4035 \or
4036 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4037 \or\or\or % vbox vtop align
4038 \or

```

```

4039      \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4040      \or\or\or\or\or\or % output math disc insert vcent mathchoice
4041      \or
4042      \aftergroup#2% 14 \beginngroup
4043      \else
4044      \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4045      \fi
4046      \fi
4047      \bbl@dirlevel\currentgrouplevel
4048      \fi
4049      #1%
4050      \fi}
4051      \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4052      \let\bbl@bodydir\@gobble
4053      \let\bbl@pagedir\@gobble
4054      \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).

```

4055      \def\bbl@xebidipar{%
4056      \let\bbl@xebidipar\relax
4057      \TeXeTstate\@ne
4058      \def\bbl@xeverypar{%
4059      \ifcase\bbl@thepardir
4060      \ifcase\bbl@thetextdir\else\beginR\fi
4061      \else
4062      {\setbox\z@\lastbox\beginR\box\z@}%
4063      \fi}%
4064      \let\bbl@severypar\everypar
4065      \newtoks\everypar
4066      \everypar=\bbl@severypar
4067      \bbl@severypar{\bbl@xeverypar\the\everypar}}
4068      \ifnum\bbl@bidimode>200
4069      \let\bbl@textdir@i\@gobbletwo
4070      \let\bbl@xebidipar\@empty
4071      \AddBabelHook{bidi}{foreign}{%
4072      \def\bbl@tempa{\def\BabelText####1}%
4073      \ifcase\bbl@thetextdir
4074      \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4075      \else
4076      \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4077      \fi}
4078      \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4079      \fi
4080      \fi

```

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

```

4081      \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4082      \AtBeginDocument{%
4083      \ifx\pdfstringdefDisableCommands\@undefined\else
4084      \ifx\pdfstringdefDisableCommands\relax\else
4085      \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4086      \fi
4087      \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`.

```

4088 \bbl@trace{Local Language Configuration}
4089 \ifx\loadlocalcfg\@undefined
4090   \@ifpackagewith{babel}{noconfigs}%
4091   {\let\loadlocalcfg\@gobble}%
4092   {\def\loadlocalcfg#1{%
4093     \InputIfFileExists{#1.cfg}%
4094     {\typeout{*****^J%
4095               * Local config file #1.cfg used^^J%
4096               *}}}%
4097   \@empty}}
4098 \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).

```

4099 \bbl@trace{Language options}
4100 \let\bbl@afterlang\relax
4101 \let\BabelModifiers\relax
4102 \let\bbl@loaded\@empty
4103 \def\bbl@load@language#1{%
4104   \InputIfFileExists{#1.ldf}%
4105   {\edef\bbl@loaded{\CurrentOption
4106     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4107     \expandafter\let\expandafter\bbl@afterlang
4108       \csname\CurrentOption.ldf-h@@k\endcsname
4109     \expandafter\let\expandafter\BabelModifiers
4110       \csname\bbl@mod@\CurrentOption\endcsname}%
4111   {\bbl@error{%
4112     Unknown option '\CurrentOption'. Either you misspelled it\\%
4113     or the language definition file \CurrentOption.ldf was not found}}%
4114     Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4115     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4116     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.

```

4117 \def\bbl@try@load@lang#1#2#3{%
4118   \IfFileExists{\CurrentOption.ldf}%
4119   {\bbl@load@language{\CurrentOption}}%
4120   {#1\bbl@load@language{#2}#3}}
4121 %
4122 \DeclareOption{hebrew}{%
4123   \input{rlbabel.def}%
4124   \bbl@load@language{hebrew}}
4125 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4126 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4127 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4128 \DeclareOption{polutonikogreek}{%
4129   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4130 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4131 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4132 \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.

```

4133 \ifx\bbl@opt@config\@nnil
4134   \@ifpackagewith{babel}{noconfigs}{}%
4135   {\InputIfFileExists{bblopts.cfg}%

```

```

4136      {\typeout{*****^J%
4137              * Local config file bblopts.cfg used^^J%
4138              *}}%
4139      {}}%
4140 \else
4141   \InputIfFileExists{\bbl@opt@config.cfg}%
4142   {\typeout{*****^J%
4143           * Local config file \bbl@opt@config.cfg used^^J%
4144           *}}%
4145   {\bbl@error{%
4146     Local config file '\bbl@opt@config.cfg' not found}{%
4147     Perhaps you misspelled it.}}%
4148 \fi

```

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

```

4149 \ifx\bbl@opt@main\@nnil
4150   \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4151     \let\bbl@tempb\@empty
4152     \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4153     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4154     \bbl@foreach\bbl@tempb{% \bbl@tempb is a reversed list
4155       \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4156         \ifodd\bbl@iniflag % = *=
4157           \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
4158         \else % n +=
4159           \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4160         \fi
4161       \fi}%
4162   \fi
4163 \else
4164   \bbl@info{Main language set with 'main='. Except if you have\\%
4165     problems, prefer the default mechanism for setting\\%
4166     the main language. Reported}%
4167 \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`).

```

4168 \ifx\bbl@opt@main\@nnil\else
4169   \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
4170   \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4171 \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.

```

4172 \bbl@foreach\bbl@language@opts{%
4173   \def\bbl@tempa{#1}%
4174   \ifx\bbl@tempa\bbl@opt@main\else
4175     \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4176       \bbl@ifunset{ds@#1}%
4177       {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4178       {}%
4179     \else % + * (other = ini)
4180       \DeclareOption{#1}{%
4181         \bbl@ldfinit
4182         \babelprovide[import]{#1}%
4183         \bbl@afterldf{}}%
4184     \fi
4185   \fi}

```

```

4186 \bbl@foreach\@classoptionslist{%
4187   \def\bbl@tempa{#1}%
4188   \ifx\bbl@tempa\bbl@opt@main\else
4189     \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4190       \bbl@ifunset{ds@#1}%
4191       {\IfFileExists{#1.ldf}%
4192        {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4193        {}}%
4194       {}%
4195     \else % + * (other = ini)
4196       \IfFileExists{babel-#1.tex}%
4197       {\DeclareOption{#1}{%
4198        \bbl@ldfinit
4199        \babelprovide[import]{#1}%
4200        \bbl@afterldf{}}}%
4201       {}}%
4202   \fi
4203 \fi}

```

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

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

```

4204 \def\AfterBabelLanguage#1{%
4205   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang{}}%
4206   \DeclareOption*{}
4207 \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.

```

4208 \bbl@trace{Option 'main'}
4209 \ifx\bbl@opt@main\@nnil
4210   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4211   \let\bbl@tempc\@empty
4212   \edef\bbl@temp1{\bbl@loaded,}
4213   \edef\bbl@temp1{\expandafter\strip@prefix\meaning\bbl@temp1}
4214   \bbl@for\bbl@tempb\bbl@tempa{%
4215     \edef\bbl@tempd{\bbl@tempb,}%
4216     \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4217     \bbl@xin@{\bbl@tempd}{\bbl@temp1}%
4218     \ifin\edef\bbl@tempc{\bbl@tempb}\fi}
4219   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4220   \expandafter\bbl@tempa\bbl@loaded,\@nnil
4221   \ifx\bbl@tempb\bbl@tempc\else
4222     \bbl@warning{%
4223       Last declared language option is '\bbl@tempc',\%
4224       but the last processed one was '\bbl@tempb'.\%
4225       The main language can't be set as both a global\%
4226       and a package option. Use 'main=\bbl@tempc' as\%
4227       option. Reported}
4228   \fi
4229 \else
4230   \ifodd\bbl@iniflag % case 1,3 (main is ini)
4231     \bbl@ldfinit
4232     \let\CurrentOption\bbl@opt@main
4233     \bbl@exp{% \bbl@opt@provide = empty if *
4234       \\\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4235     \bbl@afterldf{}
4236     \DeclareOption{\bbl@opt@main}{}
4237   \else % case 0,2 (main is ldf)

```

```

4238 \ifx\bbl@loadmain\relax
4239 \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4240 \else
4241 \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4242 \fi
4243 \ExecuteOptions{\bbl@opt@main}
4244 \@namedef{ds@\bbl@opt@main}{}%
4245 \fi
4246 \DeclareOption*{}
4247 \ProcessOptions*
4248 \fi
4249 \def\AfterBabelLanguage{%
4250 \bbl@error
4251 {Too late for \string\AfterBabelLanguage}%
4252 {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.

```

4253 \ifx\bbl@main@language\undefined
4254 \bbl@info{%
4255 You haven't specified a language as a class or package\\%
4256 option. I'll load 'nil'. Reported}
4257 \bbl@load@language{nil}
4258 \fi
4259 \</package>

```

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

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

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

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

A proxy file for `switch.def`

```

4260 \<*kernel>
4261 \let\bbl@onlyswitch\@empty
4262 \input babel.def
4263 \let\bbl@onlyswitch\@undefined
4264 \</kernel>
4265 \<*patterns>

```

## 10 Loading hyphenation patterns

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

```

4266 \<Make sure ProvidesFile is defined>
4267 \ProvidesFile{hyphen.cfg}[\<<date>> \<<version>> Babel hyphens]
4268 \xdef\bbl@format{\jobname}
4269 \def\bbl@version{\<<version>>}
4270 \def\bbl@date{\<<date>>}
4271 \ifx\AtBeginDocument\@undefined
4272 \def\@empty{}
4273 \fi
4274 \<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.

```
4275 \def\process@line#1#2 #3 #4 {%
4276   \ifx=#1%
4277     \process@synonym{#2}%
4278   \else
4279     \process@language{#1#2}{#3}{#4}%
4280   \fi
4281   \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.

```
4282 \toks@{}
4283 \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.

```
4284 \def\process@synonym#1{%
4285   \ifnum\last@language=\m@ne
4286     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4287   \else
4288     \expandafter\chardef\csname l@#1\endcsname\last@language
4289     \wlog{\string\l@#1=\string\language\the\last@language}%
4290     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4291       \csname\language\hyphenmins\endcsname
4292     \let\bbl@elt\relax
4293     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4294   \fi}
```

`\process@language` The macro `\process@language` is used to process a non-empty line from the ‘configuration file’. It has three arguments, each delimited by white space. The first argument is the ‘name’ of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions.

The first thing to do is call `\addlanguage` to allocate a pattern register and to make that register ‘active’. Then the pattern file is read.

For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file `language.dat` by adding for instance ‘:T1’ to the name of the language.

The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

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

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

Then we globally store the settings of `\lefthyphenmin` and `\righthyphenmin` and close the group.

When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

`\bbl@languages` saves a snapshot of the loaded languages in the form

`\bbl@elt{<language-name>}{<number>}{<patterns-file>}{<exceptions-file>}`. Note the last 2

arguments are empty in ‘dialects’ defined in `language.dat` with `=`. Note also the language name can have encoding info.

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

```
4295 \def\process@language#1#2#3{%
4296   \expandafter\addlanguage\csname l@#1\endcsname
```



```

4297 \expandafter\language\csname l@#1\endcsname
4298 \edef\language{#1}%
4299 \bbl@hook@everylanguage{#1}%
4300 % > luatex
4301 \bbl@get@enc#1::\@@@
4302 \begingroup
4303   \lefthyphenmin\m@ne
4304   \bbl@hook@loadpatterns{#2}%
4305   % > luatex
4306   \ifnum\lefthyphenmin=\m@ne
4307   \else
4308     \expandafter\xdef\csname #1hyphenmins\endcsname{%
4309       \the\lefthyphenmin\the\righthyphenmin}%
4310   \fi
4311 \endgroup
4312 \def\bbl@tempa{#3}%
4313 \ifx\bbl@tempa\@empty\else
4314   \bbl@hook@loadexceptions{#3}%
4315   % > luatex
4316 \fi
4317 \let\bbl@elt\relax
4318 \edef\bbl@languages{%
4319   \bbl@languages\bbl@elt{#1}\the\language}{#2}{\bbl@tempa}}%
4320 \ifnum\the\language=\z@
4321   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4322     \set@hyphenmins\tw@\thr@@\relax
4323   \else
4324     \expandafter\expandafter\expandafter\set@hyphenmins
4325       \csname #1hyphenmins\endcsname
4326   \fi
4327   \the\toks@
4328   \toks@{}%
4329 \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.

```

4330 \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.

```

4331 \def\bbl@hook@everylanguage#1{}
4332 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4333 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4334 \def\bbl@hook@loadkernel#1{%
4335   \def\addlanguage{\csname newlanguage\endcsname}%
4336   \def\adddialect##1##2{%
4337     \global\chardef##1##2\relax
4338     \wlog{\string##1 = a dialect from \string\language##2}}%
4339   \def\iflanguage##1{%
4340     \expandafter\ifx\csname l@##1\endcsname\relax
4341       \nolannerr{##1}%
4342     \else
4343       \ifnum\csname l@##1\endcsname=\language
4344         \expandafter\expandafter\expandafter\@firstoftwo
4345       \else
4346         \expandafter\expandafter\expandafter\@secondoftwo
4347       \fi
4348     \fi}%
4349   \def\providehyphenmins##1##2{%
4350     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4351       \namedef{##1hyphenmins}{##2}%
4352     \fi}%

```

```

4353 \def\set@hyphenmins##1##2{%
4354 \lefthyphenmin##1\relax
4355 \righthyphenmin##2\relax}%
4356 \def\selectlanguage{%
4357 \errhelp{Selecting a language requires a package supporting it}%
4358 \errmessage{Not loaded}}%
4359 \let\foreignlanguage\selectlanguage
4360 \let\otherlanguage\selectlanguage
4361 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4362 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4363 \def\setlocale{%
4364 \errhelp{Find an armchair, sit down and wait}%
4365 \errmessage{Not yet available}}%
4366 \let\uselocale\setlocale
4367 \let\locale\setlocale
4368 \let\selectlocale\setlocale
4369 \let\localename\setlocale
4370 \let\textlocale\setlocale
4371 \let\textlanguage\setlocale
4372 \let\languagetext\setlocale}
4373 \begingroup
4374 \def\AddBabelHook#1#2{%
4375 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4376 \def\next{\toks1}%
4377 \else
4378 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4379 \fi
4380 \next}
4381 \ifx\directlua\undefined
4382 \ifx\XeTeXinputencoding\undefined\else
4383 \input xebabel.def
4384 \fi
4385 \else
4386 \input luababel.def
4387 \fi
4388 \openin1 = babel-\bbl@format.cfg
4389 \ifeof1
4390 \else
4391 \input babel-\bbl@format.cfg\relax
4392 \fi
4393 \closein1
4394 \endgroup
4395 \bbl@hook@loadkernel{switch.def}

```

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

```
4396 \openin1 = language.dat
```

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

```

4397 \def\languagename{english}%
4398 \ifeof1
4399 \message{I couldn't find the file language.dat,\space
4400 I will try the file hyphen.tex}
4401 \input hyphen.tex\relax
4402 \chardef\l@english\z@
4403 \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.

```
4404 \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.

```
4405 \loop
4406 \endlinechar\m@ne
4407 \read1 to \bbl@line
4408 \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.

```
4409 \if T\ifeof1F\fi T\relax
4410 \ifx\bbl@line\@empty\else
4411 \edef\bbl@line{\bbl@line\space\space\space}%
4412 \expandafter\process@line\bbl@line\relax
4413 \fi
4414 \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.

```
4415 \begingroup
4416 \def\bbl@elt#1#2#3#4{%
4417 \global\language=#2\relax
4418 \gdef\language#1}%
4419 \def\bbl@elt##1##2##3##4{}}%
4420 \bbl@languages
4421 \endgroup
4422 \fi
4423 \closein1
```

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

```
4424 \if/\the\toks@\else
4425 \errhelp{language.dat loads no language, only synonyms}
4426 \errmessage{Orphan language synonym}
4427 \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.

```
4428 \let\bbl@line\@undefined
4429 \let\process@line\@undefined
4430 \let\process@synonym\@undefined
4431 \let\process@language\@undefined
4432 \let\bbl@get@enc\@undefined
4433 \let\bbl@hyph@enc\@undefined
4434 \let\bbl@tempa\@undefined
4435 \let\bbl@hook@loadkernel\@undefined
4436 \let\bbl@hook@everylanguage\@undefined
4437 \let\bbl@hook@loadpatterns\@undefined
4438 \let\bbl@hook@loadexceptions\@undefined
4439 \patterns
```

Here the code for iniTeX 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].

```
4440 \langle *More package options \rangle \equiv
4441 \chardef\bbl@bidimode\z@
4442 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4443 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
```

```

4444 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4445 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4446 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4447 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4448 <\/More package options>

```

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

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

```

4449 <(*Font selection)> ≡
4450 \bbl@trace{Font handling with fontspec}
4451 \ifx\ExplSyntaxOn\@undefined\else
4452   \def\bbl@fs@warn@nx#1#2{% \bbl@tempfs is the original macro
4453     \in@{, #1,}{, no-script, language-not-exist,}%
4454     \ifin\else\bbl@tempfs@nx{#1}{#2}\fi}
4455   \def\bbl@fs@warn@nxx#1#2#3{%
4456     \in@{, #1,}{, no-script, language-not-exist,}%
4457     \ifin\else\bbl@tempfs@nxx{#1}{#2}{#3}\fi}
4458   \def\bbl@loadfontspec{%
4459     \let\bbl@loadfontspec\relax
4460     \ifx\fontspec\@undefined
4461       \usepackage{fontspec}%
4462     \fi}%
4463 \fi
4464 \@onlypreamble\babelfont
4465 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4466   \bbl@foreach{#1}{%
4467     \expandafter\ifx\csname date##1\endcsname\relax
4468       \IfFileExists{babel-##1.tex}%
4469       {\babelprovide{##1}}%
4470     }%
4471   \fi}%
4472 \edef\bbl@tempa{#1}%
4473 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4474 \bbl@loadfontspec
4475 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4476 \bbl@bblfont}
4477 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4478   \bbl@ifunset{\bbl@tempb family}%
4479   {\bbl@providefam{\bbl@tempb}}%
4480   {}%
4481   % For the default font, just in case:
4482   \bbl@ifunset{\bbl@lsys\languagename}{\bbl@provide@lsys{\languagename}}{}%
4483   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4484   {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
4485   \bbl@exp{%
4486     \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
4487     \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
4488     \<\bbl@tempb default>\<\bbl@tempb family>}}%
4489   {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4490     \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:

```

4491 \def\bbl@providefam#1{%
4492   \bbl@exp{%
4493     \\\newcommand\<#1default>{}% Just define it
4494     \\\bbl@add@list\\bbl@font@fams{#1}%
4495     \\\DeclareRobustCommand\<#1family>{%
4496       \\\not@math@alphabet\<#1family>\relax
4497       % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4498       \\\fontfamily\<#1default>%

```

```

4499 \<ifx>\\UseHooks\\@undefined\<else>\\UseHook{#1family}\<fi>%
4500 \\selectfont}%
4501 \\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.

```

4502 \def\bbl@nostdfont#1{%
4503   \bbl@ifunset{bbl@WFF@\f@family}%
4504   {\bbl@csarg\gdef{WFF@\f@family}}}% Flag, to avoid dupl warns
4505   \bbl@infowarn{The current font is not a babel standard family:\\%
4506     #1%
4507     \fontname\font\\%
4508     There is nothing intrinsically wrong with this warning, and\\%
4509     you can ignore it altogether if you do not need these\\%
4510     families. But if they are used in the document, you should be\\%
4511     aware 'babel' will not set Script and Language for them, so\\%
4512     you may consider defining a new family with \string\babelfont.\\%
4513     See the manual for further details about \string\babelfont.\\%
4514     Reported}}
4515   }%
4516 \gdef\bbl@switchfont{%
4517   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}}%
4518   \bbl@exp{% eg Arabic -> arabic
4519     \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}%
4520   \bbl@foreach\bbl@font@fams{%
4521     \bbl@ifunset{bbl@##1dflt@\languagename}% (1) language?
4522     {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}% (2) from script?
4523       {\bbl@ifunset{bbl@##1dflt@}% 2=F - (3) from generic?
4524         {}% 123=F - nothing!
4525         {\bbl@exp{% 3=T - from generic
4526           \global\let\<bbl@##1dflt@\languagename>%
4527             \<bbl@##1dflt@>}}}%
4528         {\bbl@exp{% 2=T - from script
4529           \global\let\<bbl@##1dflt@\languagename>%
4530             \<bbl@##1dflt@*\bbl@tempa>}}}%
4531         {}% 1=T - language, already defined
4532       \def\bbl@tempa{\bbl@nostdfont}}}% TODO. Don't use \bbl@tempa
4533   \bbl@foreach\bbl@font@fams{% don't gather with prev for
4534     \bbl@ifunset{bbl@##1dflt@\languagename}%
4535     {\bbl@cs{famrst@##1}%
4536     \global\bbl@csarg\let{famrst@##1}\relax}%
4537     {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4538       \\bbl@add\\originalTeX{%
4539         \\bbl@font@rst{\bbl@cl{##1dflt}}}%
4540         \<##1default>\<##1family>{##1}}}%
4541       \\bbl@font@set\<bbl@##1dflt@\languagename>% the main part!
4542       \<##1default>\<##1family>}}}%
4543   \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.

```

4544 \ifx\f@family\undefined\else % if latex
4545   \ifcase\bbl@engine % if pdftex
4546     \let\bbl@cckckstdfonts\relax
4547   \else
4548     \def\bbl@cckckstdfonts{%
4549       \begingroup
4550       \global\let\bbl@cckckstdfonts\relax
4551       \let\bbl@tempa\@empty
4552       \bbl@foreach\bbl@font@fams{%
4553         \bbl@ifunset{bbl@##1dflt@}%
4554         {\@nameuse{##1family}%
4555         \bbl@csarg\gdef{WFF@\f@family}}}% Flag
4556         \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= \f@family\\}%

```

```

4557         \space\space\fontname\font\\\}%
4558         \bbl@csarg\xdef{##1dflt@}{\f@family}%
4559         \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4560     {}}%
4561     \ifx\bbl@tempa\@empty\else
4562         \bbl@infowarn{The following font families will use the default\\%
4563         settings for all or some languages:\\%
4564         \bbl@tempa
4565         There is nothing intrinsically wrong with it, but\\%
4566         'babel' will no set Script and Language, which could\\%
4567         be relevant in some languages. If your document uses\\%
4568         these families, consider redefining them with \string\babelfont.\\%
4569         Reported}%
4570     \fi
4571 \endgroup}
4572 \fi
4573 \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.

```

4574 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4575   \bbl@xin@{<>}{#1}%
4576   \ifin@
4577     \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4578   \fi
4579   \bbl@exp{%
4580     \def\\#2{#1}%          eg, \rmdefault{\bbl@rmdflt@lang}
4581     \\bbl@ifsamestring{#2}{\f@family}%
4582     {\\#3%
4583       \\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}}}%
4584     \let\\bbl@tempa\relax}%
4585   {}}
4586 %   TODO - next should be global?, but even local does its job. I'm
4587 %   still not sure -- must investigate:
4588 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4589   \let\bbl@tempa\bbl@mapselect
4590   \let\bbl@mapselect\relax
4591   \let\bbl@temp@fam#4%      eg, '\rmfamily', to be restored below
4592   \let#4\@empty           %   Make sure \renewfontfamily is valid
4593   \bbl@exp{%
4594     \let\\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
4595     \<keys_if_exist:nnF>{\fontspec-opentype}{Script/\bbl@cl{sname}}}%
4596     {\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4597     \<keys_if_exist:nnF>{\fontspec-opentype}{Language/\bbl@cl{lname}}}%
4598     {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4599     \let\\bbl@tempfs@nx<__fontspec_warning:nx>%
4600     \let<__fontspec_warning:nx>\\bbl@fs@warn@nx
4601     \let\\bbl@tempfs@nxx<__fontspec_warning:nxx>%
4602     \let<__fontspec_warning:nxx>\\bbl@fs@warn@nxx
4603     \\renewfontfamily\\#4%
4604     [\bbl@cl{lsys},#2]{#3}% ie \bbl@exp{..}{#3}
4605   \bbl@exp{%
4606     \let<__fontspec_warning:nx>\\bbl@tempfs@nx
4607     \let<__fontspec_warning:nxx>\\bbl@tempfs@nxx}%
4608   \begingroup
4609     #4%
4610     \xdef#1{\f@family}%      eg, \bbl@rmdflt@lang{FreeSerif(0)}
4611   \endgroup
4612   \let#4\bbl@temp@fam
4613   \bbl@exp{\let<\bbl@stripslash#4\space>}\bbl@temp@pfam
4614   \let\bbl@mapselect\bbl@tempa}%

```

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.

```
4615 \def\bb1@font@rst#1#2#3#4{%
4616   \bb1@csarg\def{famrst@#4}{\bb1@font@set{#1}#2#3}}
```

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

```
4617 \def\bb1@font@fams{rm,sf,tt}
4618 <</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.

```
4619 <<{*Footnote changes}>> ≡
4620 \bb1@trace{Bidi footnotes}
4621 \ifnum\bb1@bidimode>\z@
4622   \def\bb1@footnote#1#2#3{%
4623     \ifnextchar[%
4624       {\bb1@footnote@o{#1}{#2}{#3}}%
4625       {\bb1@footnote@x{#1}{#2}{#3}}}
4626   \long\def\bb1@footnote@x#1#2#3#4{%
4627     \bgroup
4628       \select@language@x{\bb1@main@language}%
4629       \bb1@fn@footnote{#2#1{\ignorespaces#4}#3}%
4630     \egroup}
4631   \long\def\bb1@footnote@o#1#2#3[#4]#5{%
4632     \bgroup
4633       \select@language@x{\bb1@main@language}%
4634       \bb1@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4635     \egroup}
4636   \def\bb1@footnotetext#1#2#3{%
4637     \@ifnextchar[%
4638       {\bb1@footnotetext@o{#1}{#2}{#3}}%
4639       {\bb1@footnotetext@x{#1}{#2}{#3}}}
4640   \long\def\bb1@footnotetext@x#1#2#3#4{%
4641     \bgroup
4642       \select@language@x{\bb1@main@language}%
4643       \bb1@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4644     \egroup}
4645   \long\def\bb1@footnotetext@o#1#2#3[#4]#5{%
4646     \bgroup
4647       \select@language@x{\bb1@main@language}%
4648       \bb1@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4649     \egroup}
4650   \def\BabelFootnote#1#2#3#4{%
4651     \ifx\bb1@fn@footnote\undefined
4652       \let\bb1@fn@footnote\footnote
4653     \fi
4654     \ifx\bb1@fn@footnotetext\undefined
4655       \let\bb1@fn@footnotetext\footnotetext
4656     \fi
4657     \bb1@ifblank{#2}%
4658       {\def#1{\bb1@footnote{\@firstofone}{#3}{#4}}
4659        \@namedef{\bb1@stripslash#1text}%
4660         {\bb1@footnotetext{\@firstofone}{#3}{#4}}}%
4661       {\def#1{\bb1@exp{\bb1@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4662        \@namedef{\bb1@stripslash#1text}%
4663         {\bb1@exp{\bb1@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4664   \fi
4665 <</Footnote changes>>
```

Now, the code.

```

4666 <*xetex>
4667 \def\BabelStringsDefault{unicode}
4668 \let\xebbl@stop\relax
4669 \AddBabelHook{xetex}{encodedcommands}{%
4670   \def\bbl@tempa{#1}%
4671   \ifx\bbl@tempa@empty
4672     \XeTeXinputencoding"bytes"%
4673   \else
4674     \XeTeXinputencoding"#1"%
4675   \fi
4676   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4677 \AddBabelHook{xetex}{stopcommands}{%
4678   \xebbl@stop
4679   \let\xebbl@stop\relax}
4680 \def\bbl@intraspace#1 #2 #3\@{%
4681   \bbl@csarg\gdef{xeisp@\language}%
4682     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4683 \def\bbl@intrapenalty#1\@{%
4684   \bbl@csarg\gdef{xeipn@\language}%
4685     {\XeTeXlinebreakpenalty #1\relax}}
4686 \def\bbl@provide@intraspace{%
4687   \bbl@xin@{/s}{/\bbl@cl{lnbrk}}%
4688   \ifin@ \else \bbl@xin@{/c}{/\bbl@cl{lnbrk}} \fi
4689   \ifin@
4690     \bbl@ifunset{bbl@intsp@\language}{}%
4691     {\expandafter\ifx\csname bbl@intsp@\language\endcsname\@empty\else
4692       \ifx\bbl@KVP@intraspace\@nnil
4693         \bbl@exp{%
4694           \bbl@intraspace\bbl@cl{intsp}\@}%
4695         \fi
4696         \ifx\bbl@KVP@intrapenalty\@nnil
4697           \bbl@intrapenalty0\@
4698         \fi
4699       \fi
4700       \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4701         \expandafter\bbl@intraspace\bbl@KVP@intraspace\@
4702       \fi
4703       \ifx\bbl@KVP@intrapenalty\@nnil\else
4704         \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
4705       \fi
4706       \bbl@exp{%
4707         % TODO. Execute only once (but redundant):
4708         \bbl@add\<extras\language>{%
4709           \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
4710           \<bbl@xeisp@\language>%
4711           \<bbl@xeipn@\language>%
4712           \bbl@toglobal\<extras\language>%
4713           \bbl@add\<noextras\language>{%
4714             \XeTeXlinebreaklocale ""}%
4715           \bbl@toglobal\<noextras\language>%
4716         \ifx\bbl@ispacesize\@undefined
4717           \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4718         \ifx\AtBeginDocument\@notprerr
4719           \expandafter\@secondoftwo % to execute right now
4720         \fi
4721         \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
4722       \fi}%
4723   \fi}
4724 \ifx\DisableBabelHook\@undefined\endinput\fi
4725 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4726 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4727 \DisableBabelHook{babel-fontspec}

```



```

4728 <(Font selection)>
4729 \def\bbl@provide@extra#1{}
4730 </xetex>

```

## 12.2 Layout

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titles, 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 pdf<sub>tex</sub> and xet<sub>ex</sub>.

```

4731 <*xetex | texxet>
4732 \providecommand\bbl@provide@intraspace{}
4733 \bbl@trace{Redefinitions for bidi layout}
4734 \def\bbl@sspre@caption{%
4735   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir}\bbl@main@language}}}}
4736 \ifx\bbl@opt@layout\@nnil\else % if layout=..
4737 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4738 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4739 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4740   \def\@hangfrom#1{%
4741     \setbox\@tempboxa\hbox{#1}}%
4742   \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4743   \noindent\box\@tempboxa}
4744 \def\raggedright{%
4745   \let\@centercr
4746   \bbl@startskip\z@skip
4747   \@rightskip\@flushglue
4748   \bbl@endskip\@rightskip
4749   \parindent\z@
4750   \parfillskip\bbl@startskip}
4751 \def\raggedleft{%
4752   \let\@centercr
4753   \bbl@startskip\@flushglue
4754   \bbl@endskip\z@skip
4755   \parindent\z@
4756   \parfillskip\bbl@endskip}
4757 \fi
4758 \IfBabelLayout{lists}
4759   {\bbl@sreplace\list
4760     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4761     \def\bbl@listleftmargin{%
4762       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4763     \ifcase\bbl@engine
4764       \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4765       \def\p@enumiii{\p@enumii}\theenumii{}
4766     \fi
4767     \bbl@sreplace\@verbatim
4768       {\leftskip\@totalleftmargin}%
4769       {\bbl@startskip\textwidth
4770         \advance\bbl@startskip-\linewidth}%
4771     \bbl@sreplace\@verbatim
4772       {\rightskip\z@skip}%
4773     {\bbl@endskip\z@skip}}%
4774   {}
4775 \IfBabelLayout{contents}
4776   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4777     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4778   {}
4779 \IfBabelLayout{columns}
4800   {\bbl@sreplace\@outputdblcol{\hb@xt\textwidth}{\bbl@outputbox}%
4801     \def\bbl@outputbox#1{%

```

```

4782 \hb@xt@\textwidth{%
4783 \hskip\columnwidth
4784 \hfil
4785 {\normalcolor\vrule \@width\columnseprule}%
4786 \hfil
4787 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4788 \hskip-\textwidth
4789 \hb@xt@\columnwidth{\box\@outputbox \hss}%
4790 \hskip\columnsep
4791 \hskip\columnwidth}}}%
4792 {}
4793 <(<Footnote changes>)>
4794 \IfBabelLayout{footnotes}%
4795 {\BabelFootnote\footnote\languagename{}}}%
4796 \BabelFootnote\localfootnote\languagename{}}}%
4797 \BabelFootnote\mainfootnote{}}}%
4798 {}

```

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

```

4799 \IfBabelLayout{counters*}%
4800 {\bbl@add\bbl@opt@layout{.counters.}%
4801 \AddToHook{shipout/before}{%
4802 \let\bbl@tempa\babelsublr
4803 \let\babelsublr\@firstofone
4804 \let\bbl@save@thepage\thepage
4805 \protected@edef\thepage{\thepage}%
4806 \let\babelsublr\bbl@tempa}%
4807 \AddToHook{shipout/after}{%
4808 \let\thepage\bbl@save@thepage}}}%
4809 \IfBabelLayout{counters}%
4810 {\let\bbl@latinarabic=\@arabic
4811 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
4812 \let\bbl@asciroman=\@roman
4813 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4814 \let\bbl@asciiRoman=\@Roman
4815 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}}%
4816 \fi % end if layout
4817 </xetex | texpet>

```

## 12.3 8-bit TeX

Which start just above, because some code is shared with xetex. Now, 8-bit specific stuff.

```

4818 <*texpet>
4819 \def\bbl@provide@extra#1{%
4820 % == auto-select encoding ==
4821 \ifx\bbl@encoding@select@off\@empty\else
4822 \bbl@ifunset{\bbl@encoding@#1}%
4823 {\def\@elt##1{,##1,}%
4824 \edef\bbl@tempe{\xandafter\@gobbletwo\@fontenc@load@list}%
4825 \count@\z@
4826 \bbl@foreach\bbl@tempe{%
4827 \def\bbl@tempd{##1}% Save last declared
4828 \advance\count@\@ne}%
4829 \ifnum\count@>\@ne
4830 \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
4831 \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
4832 \bbl@replace\bbl@tempa{ },}%
4833 \global\bbl@csarg\let{encoding@#1}\@empty
4834 \bbl@xin@{\bbl@tempd},{,\bbl@tempa,}%
4835 \ifin\else % if main encoding included in ini, do nothing
4836 \let\bbl@tempb\relax
4837 \bbl@foreach\bbl@tempa{%

```

```

4838         \ifx\bbbl@tempb\relax
4839             \bbbl@xin@{,##1,}{,\bbbl@tempe,}%
4840             \ifin@def\bbbl@tempb{##1}\fi
4841             \fi}%
4842         \ifx\bbbl@tempb\relax\else
4843             \bbbl@exp{%
4844                 \global\<bbbl@add>\<bbbl@preextras@#1>{\<bbbl@encoding@#1>}%
4845                 \gdef\<bbbl@encoding@#1>{%
4846                     \\babel@save\\f@encoding
4847                     \\bbbl@add\\originalTeX{\\selectfont}%
4848                     \\fontencoding{\bbbl@tempb}%
4849                     \\selectfont}}%
4850             \fi
4851             \fi
4852             \fi}%
4853             {}%
4854         \fi}
4855 \</texxet>

```

## 12.4 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, \bbbl@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).

```

4856 \<!luatex>
4857 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4858 \bbbl@trace{Read language.dat}
4859 \ifx\bbbl@readstream\@undefined
4860     \csname newread\endcsname\bbbl@readstream
4861 \fi
4862 \begingroup
4863     \toks@{}
4864     \count@% 0=start, 1=0th, 2=normal
4865     \def\bbbl@process@line#1#2 #3 #4 {%

```

```

4866 \ifx=#1%
4867 \bbl@process@synonym{#2}%
4868 \else
4869 \bbl@process@language{#1#2}{#3}{#4}%
4870 \fi
4871 \ignorespaces}
4872 \def\bbl@manylang{%
4873 \ifnum\bbl@last>\@ne
4874 \bbl@info{Non-standard hyphenation setup}%
4875 \fi
4876 \let\bbl@manylang\relax}
4877 \def\bbl@process@language#1#2#3{%
4878 \ifcase\count@
4879 \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4880 \or
4881 \count@\tw@
4882 \fi
4883 \ifnum\count@=\tw@
4884 \expandafter\addlanguage\csname l@#1\endcsname
4885 \language\allocationnumber
4886 \chardef\bbl@last\allocationnumber
4887 \bbl@manylang
4888 \let\bbl@elt\relax
4889 \xdef\bbl@languages{%
4890 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4891 \fi
4892 \the\toks@
4893 \toks@{}}
4894 \def\bbl@process@synonym@aux#1#2{%
4895 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4896 \let\bbl@elt\relax
4897 \xdef\bbl@languages{%
4898 \bbl@languages\bbl@elt{#1}{#2}{}}}%
4899 \def\bbl@process@synonym#1{%
4900 \ifcase\count@
4901 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4902 \or
4903 \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{%
4904 \else
4905 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4906 \fi}
4907 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4908 \chardef\l@english\z@
4909 \chardef\l@USenglish\z@
4910 \chardef\bbl@last\z@
4911 \global\@namedef{bbl@hyphendata@0}{\hyphen.tex}}
4912 \gdef\bbl@languages{%
4913 \bbl@elt{english}{0}{\hyphen.tex}}%
4914 \bbl@elt{USenglish}{0}{}}
4915 \else
4916 \global\let\bbl@languages@format\bbl@languages
4917 \def\bbl@elt#1#2#3#4{% Remove all except language 0
4918 \ifnum#2>\z@\else
4919 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4920 \fi}%
4921 \xdef\bbl@languages{\bbl@languages}%
4922 \fi
4923 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}} % Define flags
4924 \bbl@languages
4925 \openin\bbl@readstream=language.dat
4926 \ifeof\bbl@readstream
4927 \bbl@warning{I couldn't find language.dat. No additional\\%
4928 patterns loaded. Reported}%

```

```

4929 \else
4930 \loop
4931 \endlinechar\m@ne
4932 \read\bb1@readstream to \bb1@line
4933 \endlinechar`\^^M
4934 \if T\ifeof\bb1@readstream F\fi T\relax
4935 \ifx\bb1@line\empty\else
4936 \edef\bb1@line{\bb1@line\space\space\space}%
4937 \expandafter\bb1@process@line\bb1@line\relax
4938 \fi
4939 \repeat
4940 \fi
4941 \endgroup
4942 \bb1@trace{Macros for reading patterns files}
4943 \def\bb1@get@enc#1:#2:#3\@@{\def\bb1@hyph@enc{#2}}
4944 \ifx\babelcatcodetablenum\undefined
4945 \ifx\newcatcodetable\undefined
4946 \def\babelcatcodetablenum{5211}
4947 \def\bb1@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4948 \else
4949 \newcatcodetable\babelcatcodetablenum
4950 \newcatcodetable\bb1@pattcodes
4951 \fi
4952 \else
4953 \def\bb1@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4954 \fi
4955 \def\bb1@luapatterns#1#2{%
4956 \bb1@get@enc#1::\@@@
4957 \setbox\z@\hbox\bgroup
4958 \begingroup
4959 \savecatcodetable\babelcatcodetablenum\relax
4960 \initcatcodetable\bb1@pattcodes\relax
4961 \catcodetable\bb1@pattcodes\relax
4962 \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4963 \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
4964 \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4965 \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4966 \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4967 \catcode`\`=12 \catcode`\'=12 \catcode`\\"=12
4968 \input #1\relax
4969 \catcodetable\babelcatcodetablenum\relax
4970 \endgroup
4971 \def\bb1@tempa{#2}%
4972 \ifx\bb1@tempa\empty\else
4973 \input #2\relax
4974 \fi
4975 \egroup}%
4976 \def\bb1@patterns@lua#1{%
4977 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4978 \csname l@#1\endcsname
4979 \edef\bb1@tempa{#1}%
4980 \else
4981 \csname l@#1:\f@encoding\endcsname
4982 \edef\bb1@tempa{#1:\f@encoding}%
4983 \fi\relax
4984 \@namedef{luatexhyphen@loaded@the\language}}}% Temp
4985 \@ifundefined{bb1@hyphendata@the\language}%
4986 {\def\bb1@elt##1##2##3##4{%
4987 \ifnum##2=\csname l@bb1@tempa\endcsname % #2=spanish, dutch:OT1...
4988 \def\bb1@tempb{##3}%
4989 \ifx\bb1@tempb\empty\else % if not a synonymous
4990 \def\bb1@tempc{##3}{##4}}}%
4991 \fi

```

```

4992      \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4993      \fi}%
4994      \bbl@languages
4995      \@ifundefined{bbl@hyphendata@the\language}%
4996      {\bbl@info{No hyphenation patterns were set for\%
4997      language '\bbl@tempa'. Reported}}%
4998      {\expandafter\expandafter\expandafter\bbl@luapatterns
4999      \csname bbl@hyphendata@the\language\endcsname}}}}
5000 \endinput\fi
5001 % Here ends \ifx\AddBabelHook\@undefined
5002 % A few lines are only read by hyphen.cfg
5003 \ifx\DisableBabelHook\@undefined
5004   \AddBabelHook{luatex}{everylanguage}{%
5005     \def\process@language##1##2##3{%
5006       \def\process@line####1####2 ####3 ####4 {}}
5007   \AddBabelHook{luatex}{loadpatterns}{%
5008     \input #1\relax
5009     \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5010     {#{1}}}}
5011   \AddBabelHook{luatex}{loadexceptions}{%
5012     \input #1\relax
5013     \def\bbl@tempb##1##2{#{##1}{#1}}%
5014     \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5015     {\expandafter\expandafter\expandafter\bbl@tempb
5016     \csname bbl@hyphendata@the\language\endcsname}}
5017 \endinput\fi
5018 % Here stops reading code for hyphen.cfg
5019 % The following is read the 2nd time it's loaded
5020 \begingroup % TODO - to a lua file
5021 \catcode`\%=12
5022 \catcode`\'=12
5023 \catcode`\%=12
5024 \catcode`\:=12
5025 \directlua{
5026   Babel = Babel or {}
5027   function Babel.bytes(line)
5028     return line:gsub("(.)",
5029       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5030   end
5031   function Babel.begin_process_input()
5032     if luatexbase and luatexbase.add_to_callback then
5033       luatexbase.add_to_callback('process_input_buffer',
5034         Babel.bytes, 'Babel.bytes')
5035     else
5036       Babel.callback = callback.find('process_input_buffer')
5037       callback.register('process_input_buffer', Babel.bytes)
5038     end
5039   end
5040   function Babel.end_process_input ()
5041     if luatexbase and luatexbase.remove_from_callback then
5042       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5043     else
5044       callback.register('process_input_buffer', Babel.callback)
5045     end
5046   end
5047   function Babel.addpatterns(pp, lg)
5048     local lg = lang.new(lg)
5049     local pats = lang.patterns(lg) or ''
5050     lang.clear_patterns(lg)
5051     for p in pp:gmatch('[^s]+') do
5052       ss = ''
5053       for i in string.utfcharacters(p:gsub('%d', '')) do
5054         ss = ss .. '%d?' .. i

```

```

5055     end
5056     ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
5057     ss = ss:gsub('%.%%d%?$', '%%.')
5058     pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5059     if n == 0 then
5060         tex.sprint(
5061             [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5062             .. p .. [[]])
5063         pats = pats .. ' ' .. p
5064     else
5065         tex.sprint(
5066             [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5067             .. p .. [[]])
5068     end
5069 end
5070 lang.patterns(lg, pats)
5071 end
5072 Babel.characters = Babel.characters or {}
5073 Babel.ranges = Babel.ranges or {}
5074 function Babel.hlist_has_bidi(head)
5075     local has_bidi = false
5076     local ranges = Babel.ranges
5077     for item in node.traverse(head) do
5078         if item.id == node.id'glyph' then
5079             local itemchar = item.char
5080             local chardata = Babel.characters[itemchar]
5081             local dir = chardata and chardata.d or nil
5082             if not dir then
5083                 for nn, et in ipairs(ranges) do
5084                     if itemchar < et[1] then
5085                         break
5086                     elseif itemchar <= et[2] then
5087                         dir = et[3]
5088                         break
5089                     end
5090                 end
5091             end
5092             if dir and (dir == 'al' or dir == 'r') then
5093                 has_bidi = true
5094             end
5095         end
5096     end
5097     return has_bidi
5098 end
5099 function Babel.set_chranges_b (script, chrng)
5100     if chrng == '' then return end
5101     texio.write('Replacing ' .. script .. ' script ranges')
5102     Babel.script_blocks[script] = {}
5103     for s, e in string.gmatch(chrng..' ', '(.-)%..(.-)%s') do
5104         table.insert(
5105             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5106     end
5107 end
5108 function Babel.discard_sublr(str)
5109     if str:find( [[\string\indexentry]] ) and
5110        str:find( [[\string\babelsublr]] ) then
5111         str = str:gsub( [[\string\babelsublr%s*(%b{})]],
5112             function(m) return m:sub(2,-2) end )
5113     end
5114     return str
5115 end
5116 }
5117 \endgroup

```

```

5118 \ifx\newattribute\@undefined\else
5119   \newattribute\bbl@attr@locale
5120   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5121   \AddBabelHook{luatex}{beforeextras}{%
5122     \setattribute\bbl@attr@locale\localeid}
5123 \fi
5124 \def\BabelStringsDefault{unicode}
5125 \let\luabbl@stop\relax
5126 \AddBabelHook{luatex}{encodedcommands}{%
5127   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5128   \ifx\bbl@tempa\bbl@tempb\else
5129     \directlua{Babel.begin_process_input()}%
5130   \def\luabbl@stop{%
5131     \directlua{Babel.end_process_input()}}%
5132   \fi}%
5133 \AddBabelHook{luatex}{stopcommands}{%
5134   \luabbl@stop
5135   \let\luabbl@stop\relax}
5136 \AddBabelHook{luatex}{patterns}{%
5137   \@ifundefined{bbl@hyphendata@the\language}%
5138   {\def\bbl@elt##1##2##3##4{%
5139     \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
5140     \def\bbl@tempb{##3}%
5141     \ifx\bbl@tempb\@empty\else % if not a synonymous
5142       \def\bbl@tempc{##3}{##4}}%
5143     \fi
5144     \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5145     \fi}%
5146   \bbl@languages
5147   \@ifundefined{bbl@hyphendata@the\language}%
5148   {\bbl@info{No hyphenation patterns were set for\the
5149     language '#2'. Reported}}%
5150   {\expandafter\expandafter\expandafter\bbl@luapatterns
5151     \csname bbl@hyphendata@the\language\endcsname}}}%
5152   \@ifundefined{bbl@patterns@}{}%
5153   \begingroup
5154     \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5155     \ifin@ \else
5156       \ifx\bbl@patterns@\@empty\else
5157         \directlua{ Babel.addpatterns(
5158           [[\bbl@patterns@]], \number\language) }%
5159       \fi
5160       \@ifundefined{bbl@patterns@#1}%
5161       \@empty
5162       {\directlua{ Babel.addpatterns(
5163         [[\space\csname bbl@patterns@#1\endcsname]],
5164         \number\language) }}%
5165       \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5166     \fi
5167   \endgroup}%
5168   \bbl@exp{%
5169     \bbl@ifunset{bbl@prehc@\languagename}{}%
5170     {\bbl@ifblank{\bbl@cs{prehc@\languagename}}{}}%
5171     {\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.

```

5172 \@onlypreamble\babelpatterns
5173 \AtEndOfPackage{%
5174   \newcommand\babelpatterns[2][\@empty]{%
5175     \ifx\bbl@patterns@\relax
5176       \let\bbl@patterns@\@empty

```



```

5177 \fi
5178 \ifx\bbbl@pttnlist\@empty\else
5179 \bbbl@warning{%
5180 You must not intermingle \string\selectlanguage\space and\%
5181 \string\babelpatterns\space or some patterns will not\%
5182 be taken into account. Reported}%
5183 \fi
5184 \ifx\@empty#1%
5185 \protected@edef\bbbl@patterns@\bbbl@patterns@\space#2}%
5186 \else
5187 \edef\bbbl@tempb{\zap@space#1 \@empty}%
5188 \bbbl@for\bbbl@tempa\bbbl@tempb{%
5189 \bbbl@fixname\bbbl@tempa
5190 \bbbl@iflanguage\bbbl@tempa{%
5191 \bbbl@csarg\protected@edef{patterns@\bbbl@tempa}{%
5192 \@ifundefined{bbbl@patterns@\bbbl@tempa}%
5193 \@empty
5194 {\csname bbl@patterns@\bbbl@tempa\endcsname\space}%
5195 #2}}}%
5196 \fi}}

```

## 12.5 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.

```

5197% TODO - to a lua file
5198\directlua{
5199 Babel = Babel or {}
5200 Babel.linebreaking = Babel.linebreaking or {}
5201 Babel.linebreaking.before = {}
5202 Babel.linebreaking.after = {}
5203 Babel.locale = {} % Free to use, indexed by \localeid
5204 function Babel.linebreaking.add_before(func, pos)
5205 tex.print([[ \noexpand\csname bbl@luahyphenate\endcsname ]])
5206 if pos == nil then
5207 table.insert(Babel.linebreaking.before, func)
5208 else
5209 table.insert(Babel.linebreaking.before, pos, func)
5210 end
5211 end
5212 function Babel.linebreaking.add_after(func)
5213 tex.print([[ \noexpand\csname bbl@luahyphenate\endcsname ]])
5214 table.insert(Babel.linebreaking.after, func)
5215 end
5216 }
5217\def\bbbl@intraspace#1 #2 #3\@{#1}%
5218 \directlua{
5219 Babel = Babel or {}
5220 Babel.intraspaces = Babel.intraspaces or {}
5221 Babel.intraspaces['\csname bbl@sbcpr@language\endcsname'] = %
5222 {b = #1, p = #2, m = #3}
5223 Babel.locale_props[\the\localeid].intraspace = %
5224 {b = #1, p = #2, m = #3}
5225 }}
5226\def\bbbl@intrapenalty#1\@{#1}%
5227 \directlua{
5228 Babel = Babel or {}
5229 Babel.intrapenalties = Babel.intrapenalties or {}
5230 Babel.intrapenalties['\csname bbl@sbcpr@language\endcsname'] = #1
5231 Babel.locale_props[\the\localeid].intrapenalty = #1
5232 }}

```

```

5233 \beginingroup
5234 \catcode`\%=12
5235 \catcode`\^=14
5236 \catcode`\'=12
5237 \catcode`\~=12
5238 \gdef\bbl@seaintraspace{^
5239   \let\bbl@seaintraspace\relax
5240   \directlua{
5241     Babel = Babel or {}
5242     Babel.sea_enabled = true
5243     Babel.sea_ranges = Babel.sea_ranges or {}
5244     function Babel.set_chranges (script, chrng)
5245       local c = 0
5246       for s, e in string.gmatch(chrng..' ', '(.-%.-%.-%s') do
5247         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5248         c = c + 1
5249       end
5250     end
5251     function Babel.sea_disc_to_space (head)
5252       local sea_ranges = Babel.sea_ranges
5253       local last_char = nil
5254       local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5255       for item in node.traverse(head) do
5256         local i = item.id
5257         if i == node.id'glyph' then
5258           last_char = item
5259         elseif i == 7 and item.subtype == 3 and last_char
5260           and last_char.char > 0x0C99 then
5261           quad = font.getfont(last_char.font).size
5262           for lg, rg in pairs(sea_ranges) do
5263             if last_char.char > rg[1] and last_char.char < rg[2] then
5264               lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyril1
5265               local intraspace = Babel.intraspaces[lg]
5266               local intrapenalty = Babel.intrapenalties[lg]
5267               local n
5268               if intrapenalty ~= 0 then
5269                 n = node.new(14, 0)    ^% penalty
5270                 n.penalty = intrapenalty
5271                 node.insert_before(head, item, n)
5272               end
5273               n = node.new(12, 13)    ^% (glue, spaceskip)
5274               node.setglue(n, intraspace.b * quad,
5275                 intraspace.p * quad,
5276                 intraspace.m * quad)
5277               node.insert_before(head, item, n)
5278               node.remove(head, item)
5279             end
5280           end
5281         end
5282       end
5283     end
5284   }^^
5285   \bbl@luahyphenate}

```

## 12.6 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.

```

5286 \catcode`\%=14

```

```

5287 \gdef\bbl@ckintraspace{%
5288   \let\bbl@ckintraspace\relax
5289   \directlua{
5290     Babel = Babel or {}
5291     require('babel-data-cjk.lua')
5292     Babel.cjk_enabled = true
5293     function Babel.cjk_linebreak(head)
5294       local GLYPH = node.id'glyph'
5295       local last_char = nil
5296       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5297       local last_class = nil
5298       local last_lang = nil
5299
5300       for item in node.traverse(head) do
5301         if item.id == GLYPH then
5302
5303           local lang = item.lang
5304
5305           local LOCALE = node.get_attribute(item,
5306             Babel.attr_locale)
5307           local props = Babel.locale_props[LOCALE]
5308
5309           local class = Babel.cjk_class[item.char].c
5310
5311           if props.cjk_quotes and props.cjk_quotes[item.char] then
5312             class = props.cjk_quotes[item.char]
5313           end
5314
5315           if class == 'cp' then class = 'cl' end % ]) as CL
5316           if class == 'id' then class = 'I' end
5317
5318           local br = 0
5319           if class and last_class and Babel.cjk_breaks[last_class][class] then
5320             br = Babel.cjk_breaks[last_class][class]
5321           end
5322
5323           if br == 1 and props.linebreak == 'c' and
5324             lang ~= \the\l@nohyphenation\space and
5325             last_lang ~= \the\l@nohyphenation then
5326             local intrapenalty = props.intrapenalty
5327             if intrapenalty ~= 0 then
5328               local n = node.new(14, 0)      % penalty
5329               n.penalty = intrapenalty
5330               node.insert_before(head, item, n)
5331             end
5332             local intraspace = props.intraspace
5333             local n = node.new(12, 13)      % (glue, spaceskip)
5334             node.setglue(n, intraspace.b * quad,
5335               intraspace.p * quad,
5336               intraspace.m * quad)
5337             node.insert_before(head, item, n)
5338           end
5339
5340           if font.getfont(item.font) then
5341             quad = font.getfont(item.font).size
5342           end
5343           last_class = class
5344           last_lang = lang
5345         else % if penalty, glue or anything else
5346           last_class = nil
5347         end
5348       end
5349       lang.hyphenate(head)

```

```

5350     end
5351 }%
5352 \bbl@luahyphenate}
5353 \gdef\bbl@luahyphenate{%
5354 \let\bbl@luahyphenate\relax
5355 \directlua{
5356   luatexbase.add_to_callback('hyphenate',
5357   function (head, tail)
5358     if Babel.linebreaking.before then
5359       for k, func in ipairs(Babel.linebreaking.before) do
5360         func(head)
5361       end
5362     end
5363     if Babel.cjk_enabled then
5364       Babel.cjk_linebreak(head)
5365     end
5366     lang.hyphenate(head)
5367     if Babel.linebreaking.after then
5368       for k, func in ipairs(Babel.linebreaking.after) do
5369         func(head)
5370       end
5371     end
5372     if Babel.sea_enabled then
5373       Babel.sea_disc_to_space(head)
5374     end
5375   end,
5376   'Babel.hyphenate')
5377 }
5378 }
5379 \endgroup
5380 \def\bbl@provide@intraspace{%
5381 \bbl@ifunset\bbl@intsp@{\languagename}{}%
5382 {\xexpandafter\ifx\cename\bbl@intsp@\languagename\endcsname\@empty\else
5383 \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
5384 \ifin@ % cjk
5385 \bbl@cjk@intraspace
5386 \directlua{
5387   Babel = Babel or {}
5388   Babel.locale_props = Babel.locale_props or {}
5389   Babel.locale_props[\the\localeid].linebreak = 'c'
5390 }%
5391 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\@}%
5392 \ifx\bbl@KVP@intrapenalty\@nnil
5393 \bbl@intrapenalty0\@
5394 \fi
5395 \else % sea
5396 \bbl@sea@intraspace
5397 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\@}%
5398 \directlua{
5399   Babel = Babel or {}
5400   Babel.sea_ranges = Babel.sea_ranges or {}
5401   Babel.set_chranges('\bbl@cl{sbcpr}',
5402   '\bbl@cl{chrng}')
5403 }%
5404 \ifx\bbl@KVP@intrapenalty\@nnil
5405 \bbl@intrapenalty0\@
5406 \fi
5407 \fi
5408 \fi
5409 \ifx\bbl@KVP@intrapenalty\@nnil\else
5410 \xexpandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
5411 \fi}}

```

## 12.7 Arabic justification

```
5412 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5413 \def\bblar@chars{%
5414   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5415   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5416   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5417 \def\bblar@elongated{%
5418   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5419   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5420   0649,064A}
5421 \begingroup
5422   \catcode\_ =11 \catcode\`:=11
5423   \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5424 \endgroup
5425 \gdef\bbl@arabicjust{%
5426   \let\bbl@arabicjust\relax
5427   \newattribute\bblar@kashida
5428   \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5429   \bblar@kashida=\z@
5430   \bbl@patchfont{\bbl@parsejalt}}%
5431   \directlua{
5432     Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5433     Babel.arabic.elong_map[\the\localeid] = {}
5434     luatexbase.add_to_callback('post_linebreak_filter',
5435       Babel.arabic.justify, 'Babel.arabic.justify')
5436     luatexbase.add_to_callback('hpack_filter',
5437       Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5438   }}%
5439 % Save both node lists to make replacement. TODO. Save also widths to
5440 % make computations
5441 \def\bblar@fetchjalt#1#2#3#4{%
5442   \bbl@exp{\bbl@foreach{#1}}{%
5443     \bbl@ifunset{bblar@JE@##1}%
5444     {\setbox\z@\hbox{^^^200d\char"##1#2}}%
5445     {\setbox\z@\hbox{^^^200d\char"@nameuse{bblar@JE@##1}#2}}%
5446   \directlua{%
5447     local last = nil
5448     for item in node.traverse(tex.box[0].head) do
5449       if item.id == node.id'glyph' and item.char > 0x600 and
5450         not (item.char == 0x200D) then
5451         last = item
5452       end
5453     end
5454     Babel.arabic.#3['##1#4'] = last.char
5455   }}%
5456 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5457 % perhaps other tables (falt?, csw?). What about kaf? And diacritic
5458 % positioning?
5459 \gdef\bbl@parsejalt{%
5460   \ifx\addfontfeature\@undefined\else
5461     \bbl@xin{/e}{\bbl@c1{lnbrk}}%
5462   \fin@
5463   \directlua{%
5464     if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5465       Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5466       tex.print([[string\cspace bbl@parsejalti\endcspace]])
5467     end
5468   }}%
5469   \fi
5470 \fi}
5471 \gdef\bbl@parsejalti{%
5472   \begingroup
```

```

5473 \let\bbl@parsejalt\relax % To avoid infinite loop
5474 \edef\bbl@tempb{\fontid\font}%
5475 \bblar@nofswarn
5476 \bblar@fetchjalt\bblar@elongated{}{from}{}%
5477 \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5478 \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5479 \addfontfeature{RawFeature+=jalt}%
5480 % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5481 \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5482 \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5483 \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5484 \directlua{%
5485     for k, v in pairs(Babel.arabic.from) do
5486         if Babel.arabic.dest[k] and
5487             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5488             Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5489                 [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5490         end
5491     end
5492 }%
5493 \endgroup}
5494 %
5495 \begingroup
5496 \catcode`#=11
5497 \catcode`~=11
5498 \directlua{
5499
5500 Babel.arabic = Babel.arabic or {}
5501 Babel.arabic.from = {}
5502 Babel.arabic.dest = {}
5503 Babel.arabic.justify_factor = 0.95
5504 Babel.arabic.justify_enabled = true
5505
5506 function Babel.arabic.justify(head)
5507     if not Babel.arabic.justify_enabled then return head end
5508     for line in node.traverse_id(node.id'hlist', head) do
5509         Babel.arabic.justify_hlist(head, line)
5510     end
5511     return head
5512 end
5513
5514 function Babel.arabic.justify_hbox(head, gc, size, pack)
5515     local has_inf = false
5516     if Babel.arabic.justify_enabled and pack == 'exactly' then
5517         for n in node.traverse_id(12, head) do
5518             if n.stretch_order > 0 then has_inf = true end
5519         end
5520         if not has_inf then
5521             Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5522         end
5523     end
5524     return head
5525 end
5526
5527 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5528     local d, new
5529     local k_list, k_item, pos_inline
5530     local width, width_new, full, k_curr, wt_pos, goal, shift
5531     local subst_done = false
5532     local elong_map = Babel.arabic.elong_map
5533     local last_line
5534     local GLYPH = node.id'glyph'
5535     local KASHIDA = Babel.attr_kashida

```

```

5536 local LOCALE = Babel.attr_locale
5537
5538 if line == nil then
5539     line = {}
5540     line.glue_sign = 1
5541     line.glue_order = 0
5542     line.head = head
5543     line.shift = 0
5544     line.width = size
5545 end
5546
5547 % Exclude last line. todo. But-- it discards one-word lines, too!
5548 % ? Look for glue = 12:15
5549 if (line.glue_sign == 1 and line.glue_order == 0) then
5550     elongs = {}      % Stores elongated candidates of each line
5551     k_list = {}      % And all letters with kashida
5552     pos_inline = 0   % Not yet used
5553
5554     for n in node.traverse_id(GLYPH, line.head) do
5555         pos_inline = pos_inline + 1 % To find where it is. Not used.
5556
5557         % Elongated glyphs
5558         if elong_map then
5559             local locale = node.get_attribute(n, LOCALE)
5560             if elong_map[locale] and elong_map[locale][n.font] and
5561                 elong_map[locale][n.font][n.char] then
5562                 table.insert(elongs, {node = n, locale = locale} )
5563                 node.set_attribute(n.prev, KASHIDA, 0)
5564             end
5565         end
5566
5567         % Tatwil
5568         if Babel.kashida_wts then
5569             local k_wt = node.get_attribute(n, KASHIDA)
5570             if k_wt > 0 then % todo. parameter for multi inserts
5571                 table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5572             end
5573         end
5574     end % of node.traverse_id
5575
5576
5577 if #elongs == 0 and #k_list == 0 then goto next_line end
5578 full = line.width
5579 shift = line.shift
5580 goal = full * Babel.arabic.justify_factor % A bit crude
5581 width = node.dimensions(line.head) % The 'natural' width
5582
5583 % == Elongated ==
5584 % Original idea taken from 'chickenize'
5585 while (#elongs > 0 and width < goal) do
5586     subst_done = true
5587     local x = #elongs
5588     local curr = elongs[x].node
5589     local oldchar = curr.char
5590     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5591     width = node.dimensions(line.head) % Check if the line is too wide
5592     % Substitute back if the line would be too wide and break:
5593     if width > goal then
5594         curr.char = oldchar
5595         break
5596     end
5597     % If continue, pop the just substituted node from the list:
5598     table.remove(elongs, x)

```

```

5599   end
5600
5601   % == Tatwil ==
5602   if #k_list == 0 then goto next_line end
5603
5604   width = node.dimensions(line.head)    % The 'natural' width
5605   k_curr = #k_list
5606   wt_pos = 1
5607
5608   while width < goal do
5609     subst_done = true
5610     k_item = k_list[k_curr].node
5611     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5612       d = node.copy(k_item)
5613       d.char = 0x0640
5614       line.head, new = node.insert_after(line.head, k_item, d)
5615       width_new = node.dimensions(line.head)
5616       if width > goal or width == width_new then
5617         node.remove(line.head, new) % Better compute before
5618         break
5619       end
5620       width = width_new
5621     end
5622     if k_curr == 1 then
5623       k_curr = #k_list
5624       wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5625     else
5626       k_curr = k_curr - 1
5627     end
5628   end
5629
5630   ::next_line::
5631
5632   % Must take into account marks and ins, see luatex manual.
5633   % Have to be executed only if there are changes. Investigate
5634   % what's going on exactly.
5635   if subst_done and not gc then
5636     d = node.hpack(line.head, full, 'exactly')
5637     d.shift = shift
5638     node.insert_before(head, line, d)
5639     node.remove(head, line)
5640   end
5641 end % if process line
5642 end
5643 }
5644 \endgroup
5645 \fi\fi % Arabic just block

```

## 12.8 Common stuff

```

5646 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5647 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ccheckstdfonts}
5648 \DisableBabelHook{babel-fontspec}
5649 <<Font selection>>

```

## 12.9 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 form 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.



```

5650% TODO - to a lua file
5651\directlua{
5652Babel.script_blocks = {
5653  ['dflt'] = {},
5654  ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5655             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5656  ['Armn'] = {{0x0530, 0x058F}},
5657  ['Beng'] = {{0x0980, 0x09FF}},
5658  ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5659  ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5660  ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5661             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5662  ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5663  ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5664             {0xAB00, 0xAB2F}},
5665  ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5666  % Don't follow strictly Unicode, which places some Coptic letters in
5667  % the 'Greek and Coptic' block
5668  ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5669  ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5670             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5671             {0xF900, 0FAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5672             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5673             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5674             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5675  ['Hebr'] = {{0x0590, 0x05FF}},
5676  ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5677             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5678  ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5679  ['Knda'] = {{0x0C80, 0x0CFF}},
5680  ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5681             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5682             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5683  ['Lao'] = {{0x0E80, 0x0EFF}},
5684  ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5685             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5686             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5687  ['Mahj'] = {{0x11150, 0x1117F}},
5688  ['Mlym'] = {{0x0D00, 0x0D7F}},
5689  ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5690  ['Orya'] = {{0x0B00, 0x0B7F}},
5691  ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5692  ['Sycr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5693  ['Taml'] = {{0x0B80, 0x0BFF}},
5694  ['Telu'] = {{0x0C00, 0x0C7F}},
5695  ['Tfng'] = {{0x2D30, 0x2D7F}},
5696  ['Thai'] = {{0x0E00, 0x0E7F}},
5697  ['Tibt'] = {{0x0F00, 0x0FFF}},
5698  ['Vaii'] = {{0xA500, 0xA63F}},
5699  ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5700}
5701
5702Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5703Babel.script_blocks.Hant = Babel.script_blocks.Hans
5704Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5705
5706function Babel.locale_map(head)
5707  if not Babel.locale_mapped then return head end
5708
5709  local LOCALE = Babel.attr_locale
5710  local GLYPH = node.id('glyph')
5711  local inmath = false
5712  local toloc_save

```

```

5713 for item in node.traverse(head) do
5714   local toloc
5715   if not inmath and item.id == GLYPH then
5716     % Optimization: build a table with the chars found
5717     if Babel.chr_to_loc[item.char] then
5718       toloc = Babel.chr_to_loc[item.char]
5719     else
5720       for lc, maps in pairs(Babel.loc_to_scr) do
5721         for _, rg in pairs(maps) do
5722           if item.char >= rg[1] and item.char <= rg[2] then
5723             Babel.chr_to_loc[item.char] = lc
5724             toloc = lc
5725             break
5726           end
5727         end
5728       end
5729     end
5730     % Now, take action, but treat composite chars in a different
5731     % fashion, because they 'inherit' the previous locale. Not yet
5732     % optimized.
5733     if not toloc and
5734       (item.char >= 0x0300 and item.char <= 0x036F) or
5735       (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5736       (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5737       toloc = toloc_save
5738     end
5739     if toloc and Babel.locale_props[toloc] and
5740       Babel.locale_props[toloc].letters and
5741       tex.getcatcode(item.char) \string~= 11 then
5742       toloc = nil
5743     end
5744     if toloc and toloc > -1 then
5745       if Babel.locale_props[toloc].lg then
5746         item.lang = Babel.locale_props[toloc].lg
5747         node.set_attribute(item, LOCALE, toloc)
5748       end
5749       if Babel.locale_props[toloc]['/'..item.font] then
5750         item.font = Babel.locale_props[toloc]['/'..item.font]
5751       end
5752       toloc_save = toloc
5753     end
5754   elseif not inmath and item.id == 7 then % Apply recursively
5755     item.replace = item.replace and Babel.locale_map(item.replace)
5756     item.pre     = item.pre and Babel.locale_map(item.pre)
5757     item.post    = item.post and Babel.locale_map(item.post)
5758   elseif item.id == node.id'math' then
5759     inmath = (item.subtype == 0)
5760   end
5761 end
5762 return head
5763 end
5764 }

```

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

```

5765 \newcommand\babelcharproperty[1]{%
5766   \count@=#1\relax
5767   \ifvmode
5768     \expandafter\babel@chprop
5769   \else
5770     \bbl@error{\string\babelcharproperty\space can be used only in\%
5771               vertical mode (preamble or between paragraphs)}%
5772     {See the manual for futher info}%

```

```

5773 \fi}
5774 \newcommand\bbl@chprop[3][\the\count@]{%
5775 \@tempcnta=#1\relax
5776 \bbl@ifunset{\bbl@chprop@#2}%
5777 {\bbl@error{No property named '#2'. Allowed values are\\%
5778 direction (bc), mirror (bmg), and linebreak (lb)}%
5779 {See the manual for futher info}}%
5780 {}%
5781 \loop
5782 \bbl@cs{chprop@#2}{#3}%
5783 \ifnum\count@<\@tempcnta
5784 \advance\count@\@ne
5785 \repeat}
5786 \def\bbl@chprop@direction#1{%
5787 \directlua{
5788 Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5789 Babel.characters[\the\count@]['d'] = '#1'
5790 }}
5791 \let\bbl@chprop@bc\bbl@chprop@direction
5792 \def\bbl@chprop@mirror#1{%
5793 \directlua{
5794 Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5795 Babel.characters[\the\count@]['m'] = '\number#1'
5796 }}
5797 \let\bbl@chprop@bmg\bbl@chprop@mirror
5798 \def\bbl@chprop@linebreak#1{%
5799 \directlua{
5800 Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5801 Babel.cjk_characters[\the\count@]['c'] = '#1'
5802 }}
5803 \let\bbl@chprop@lb\bbl@chprop@linebreak
5804 \def\bbl@chprop@locale#1{%
5805 \directlua{
5806 Babel.chr_to_loc = Babel.chr_to_loc or {}
5807 Babel.chr_to_loc[\the\count@] =
5808 \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
5809 }}

```

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.

```

5810 \directlua{
5811 Babel.nohyphenation = \the\l@nohyphenation
5812 }

```

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

```

5813 \begingroup
5814 \catcode`\~ =12
5815 \catcode`\ =12
5816 \catcode`\& =14
5817 \catcode`\| =12
5818 \gdef\babelprehyphenation{&%
5819 \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}}
5820 \gdef\babelposthyphenation{&%
5821 \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}}
5822 \gdef\bbl@postlinebreak{\bbl@settransform{2}[]} &% WIP
5823 \gdef\bbl@settransform#1[#2]#3#4#5{&%

```

```

5824 \ifcase#1
5825 \bbl@activateprehyphen
5826 \or
5827 \bbl@activateposthyphen
5828 \fi
5829 \beginingroup
5830 \def\babeltempa{\bbl@add@list\babeltempb}&%
5831 \let\babeltempb\empty
5832 \def\bbl@tempa{#5}&%
5833 \bbl@replace\bbl@tempa{,}{,}&% TODO. Ugly trick to preserve {}
5834 \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5835 \bbl@ifsamestring{##1}{remove}&%
5836 {\bbl@add@list\babeltempb{nil}}&%
5837 {\directlua{
5838 local rep = [=##1]=]
5839 rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5840 rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5841 rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5842 if #1 == 0 or #1 == 2 then
5843 rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5844 'space = {' .. '%2, %3, %4' .. '}')
5845 rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5846 'spacefactor = {' .. '%2, %3, %4' .. '}')
5847 rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
5848 else
5849 rep = rep:gsub('(no)%s*=%s*([^\s,]*)', Babel.capture_func)
5850 rep = rep:gsub('(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5851 rep = rep:gsub('(post)%s*=%s*([^\s,]*)', Babel.capture_func)
5852 end
5853 tex.print([[\\string\babeltempa{}} .. rep .. [{}]])
5854 }}&%
5855 \bbl@foreach\babeltempb{&%
5856 \bbl@forkv{##1}{&%
5857 \in@{,###1,}{,nil,step,data,remove,insert,string,no,pre,&%
5858 no,post,penalty,kashida,space,spacefactor,}&%
5859 \ifin@\\else
5860 \bbl@error
5861 {Bad option '###1' in a transform.\\&%
5862 I'll ignore it but expect more errors}&%
5863 {See the manual for further info.}&%
5864 \fi}}&%
5865 \let\bbl@kv@attribute\relax
5866 \let\bbl@kv@label\relax
5867 \let\bbl@kv@fonts\empty
5868 \bbl@forkv{#2}{\bbl@csarg\edef{kv##1}{##2}}&%
5869 \ifx\bbl@kv@fonts\empty\\else\bbl@settransfont\\fi
5870 \ifx\bbl@kv@attribute\relax
5871 \ifx\bbl@kv@label\relax\\else
5872 \bbl@exp{\\bbl@trim@def\\bbl@kv@fonts{\bbl@kv@fonts}}&%
5873 \bbl@replace\bbl@kv@fonts{,}{,}&%
5874 \edef\bbl@kv@attribute{\bbl@ATR@\\bbl@kv@label @#3@\\bbl@kv@fonts}&%
5875 \count@\\z@
5876 \def\bbl@elt##1##2##3{&%
5877 \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
5878 {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
5879 {\count@\\ne}&%
5880 {\bbl@error
5881 {Transforms cannot be re-assigned to different\\&%
5882 fonts. The conflict is in '\\bbl@kv@label'.\\&%
5883 Apply the same fonts or use a different label}&%
5884 {See the manual for further details.}}}&%
5885 }}&%
5886 \bbl@transfont@list

```

```

5887     \ifnum\count@=\z@
5888         \bbl@exp{\global\\bbl@add\\bbl@transfont@list
5889             {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}%&
5890     \fi
5891     \bbl@ifunset{\bbl@kv@attribute}%&
5892     {\global\bbl@carg\newattribute{\bbl@kv@attribute}}%&
5893     {}%&
5894     \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
5895 \fi
5896 \else
5897     \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}%&
5898 \fi
5899 \directlua{
5900     local lbkr = Babel.linebreaking.replacements[#1]
5901     local u = unicode.utf8
5902     local id, attr, label
5903     if #1 == 0 or #1 == 2 then
5904         id = \the\csname bbl@id@#3\endcsname\space
5905     else
5906         id = \the\csname l@#3\endcsname\space
5907     end
5908     \ifx\bbl@kv@attribute\relax
5909         attr = -1
5910     \else
5911         attr = luatexbase.registernumber'\bbl@kv@attribute'
5912     \fi
5913     \ifx\bbl@kv@label\relax\else &% Same refs:
5914         label = [==[\bbl@kv@label]==]
5915     \fi
5916     &% Convert pattern:
5917     local patt = string.gsub([==[#4]==], '%s', '')
5918     if #1 == 0 or #1 == 2 then
5919         patt = string.gsub(patt, '|', ' ')
5920     end
5921     if not u.find(patt, '()', nil, true) then
5922         patt = '()' .. patt .. '()'
5923     end
5924     if #1 == 1 then
5925         patt = string.gsub(patt, '%(%)^', '^()')
5926         patt = string.gsub(patt, '%$$(%)', '()$')
5927     end
5928     patt = u.gsub(patt, '{(.)}',
5929         function (n)
5930             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5931         end)
5932     patt = u.gsub(patt, '{(%x%x%x%x+)}',
5933         function (n)
5934             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
5935         end)
5936     lbkr[id] = lbkr[id] or {}
5937     table.insert(lbkr[id],
5938         { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
5939 }%&
5940 \endgroup}
5941 \endgroup
5942 \let\bbl@transfont@list\@empty
5943 \def\bbl@settransfont{%
5944     \global\let\bbl@settransfont\relax % Execute only once
5945     \gdef\bbl@transfont{%
5946         \def\bbl@elt####1####2####3{%
5947             \bbl@ifblank{####3}%
5948             {\count@=tw@}% Do nothing if no fonts
5949             {\count@=\z@

```

```

5950 \bbl@vforeach{####3}{%
5951 \def\bbl@tempd{#####1}%
5952 \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
5953 \ifx\bbl@tempd\bbl@tempe
5954 \count@ \@ne
5955 \else\ifx\bbl@tempd\bbl@transfam
5956 \count@ \@ne
5957 \fi\fi}%
5958 \ifcase\count@
5959 \bbl@csarg\unsetattribute{ATR@###2@###1@###3}%
5960 \or
5961 \bbl@csarg\setattribute{ATR@###2@###1@###3}\@ne
5962 \fi}}%
5963 \bbl@transfont@list}%
5964 \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
5965 \gdef\bbl@transfam{-unknown-}%
5966 \bbl@foreach\bbl@font@fams{%
5967 \AddToHook{##1family}{\def\bbl@transfam{##1}}%
5968 \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
5969 {\xdef\bbl@transfam{##1}}%
5970 {}}
5971 \DeclareRobustCommand\enablelocaletransform[1]{%
5972 \bbl@ifunset{\bbl@ATR@#1@languagename @}%
5973 {\bbl@error
5974 {'#1' for '\languagename' cannot be enabled.\\%
5975 Maybe there is a typo or it's a font-dependent transform}%
5976 {See the manual for further details.}}%
5977 {\bbl@csarg\setattribute{ATR@#1@languagename @}\@ne}}
5978 \DeclareRobustCommand\disablelocaletransform[1]{%
5979 \bbl@ifunset{\bbl@ATR@#1@languagename @}%
5980 {\bbl@error
5981 {'#1' for '\languagename' cannot be disabled.\\%
5982 Maybe there is a typo or it's a font-dependent transform}%
5983 {See the manual for further details.}}%
5984 {\bbl@csarg\unsetattribute{ATR@#1@languagename @}}}
5985 \def\bbl@activateposthyphen{%
5986 \let\bbl@activateposthyphen\relax
5987 \directlua{
5988 require('babel-transforms.lua')
5989 Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5990 }}
5991 \def\bbl@activateprehyphen{%
5992 \let\bbl@activateprehyphen\relax
5993 \directlua{
5994 require('babel-transforms.lua')
5995 Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5996 }}

```

## 12.10 Bidi

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

```

5997 \def\bbl@activate@preotf{%
5998 \let\bbl@activate@preotf\relax % only once
5999 \directlua{
6000 Babel = Babel or {}
6001 %
6002 function Babel.pre_otfload_v(head)
6003 if Babel.numbers and Babel.digits_mapped then
6004 head = Babel.numbers(head)
6005 end
6006 if Babel.bidi_enabled then

```

```

6007     head = Babel.bidi(head, false, dir)
6008 end
6009 return head
6010 end
6011 %
6012 function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6013   if Babel.numbers and Babel.digits_mapped then
6014     head = Babel.numbers(head)
6015   end
6016   if Babel.bidi_enabled then
6017     head = Babel.bidi(head, false, dir)
6018   end
6019   return head
6020 end
6021 %
6022 luatexbase.add_to_callback('pre_linebreak_filter',
6023   Babel.pre_otfload_v,
6024   'Babel.pre_otfload_v',
6025   luatexbase.priority_in_callback('pre_linebreak_filter',
6026     'luaotfload.node_processor') or nil)
6027 %
6028 luatexbase.add_to_callback('hpack_filter',
6029   Babel.pre_otfload_h,
6030   'Babel.pre_otfload_h',
6031   luatexbase.priority_in_callback('hpack_filter',
6032     'luaotfload.node_processor') or nil)
6033 }}

```

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=`.

```

6034 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
6035   \let\bbl@beforeforeign\leavevmode
6036   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6037   \RequirePackage{luatexbase}
6038   \bbl@activate@preotf
6039   \directlua{
6040     require('babel-data-bidi.lua')
6041     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6042       require('babel-bidi-basic.lua')
6043     \or
6044       require('babel-bidi-basic-r.lua')
6045     \fi}
6046 % TODO - to locale_props, not as separate attribute
6047 \newattribute\bbl@attr@dir
6048 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6049 % TODO. I don't like it, hackish:
6050 \bbl@exp{\output{\bodydir\pagedir\the\output}}
6051 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6052 \fi\fi
6053 \chardef\bbl@thetextdir\z@
6054 \chardef\bbl@thepardir\z@
6055 \def\bbl@getluadir#1{%
6056   \directlua{
6057     if tex.#1dir == 'TLT' then
6058       tex.sprint('0')
6059     elseif tex.#1dir == 'TRT' then
6060       tex.sprint('1')
6061     end}}
6062 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6063   \ifcase#3\relax
6064     \ifcase\bbl@getluadir{#1}\relax\else
6065       #2 TLT\relax

```

```

6066 \fi
6067 \else
6068 \ifcase\bbl@getluadir{#1}\relax
6069 #2 TRT\relax
6070 \fi
6071 \fi}
6072 \def\bbl@thedir{0}
6073 \def\bbl@textdir#1{%
6074 \bbl@setluadir{text}\textdir{#1}%
6075 \chardef\bbl@thetextdir#1\relax
6076 \edef\bbl@thedir{\the\numexpr\bbl@thepardir*3+#1}%
6077 \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
6078 \def\bbl@pardir#1{%
6079 \bbl@setluadir{par}\pardir{#1}%
6080 \chardef\bbl@thepardir#1\relax}
6081 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
6082 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
6083 \def\bbl@dirparastext{\pardir\the\textdir\relax}% %%%
6084 %
6085 \ifnum\bbl@bidimode>\z@
6086 \def\bbl@insidemath{0}%
6087 \def\bbl@mathboxdir{%
6088 \ifcase\bbl@thetextdir\relax
6089 \everyhbox{\bbl@mathboxdir@aux L}%
6090 \else
6091 \everyhbox{\bbl@mathboxdir@aux R}%
6092 \fi}
6093 \def\bbl@mathboxdir@aux#1{%
6094 \@ifnextchar\egroup{}}{\textdir T#1T\relax}}
6095 \def\bbl@everymath{%
6096 \bbl@mathboxdir
6097 \def\bbl@insidemath{1}}
6098 \def\bbl@everydisplay{%
6099 \bbl@mathboxdir
6100 \def\bbl@everymath{\bbl@mathboxdir}%
6101 \def\bbl@insidemath{2}}
6102 \frozen@everymath\expandafter{%
6103 \expandafter\bbl@everymath\the\frozen@everymath}
6104 \frozen@everydisplay\expandafter{%
6105 \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6106 \AtBeginDocument{
6107 \directlua{
6108 Babel.use_math_box_dir = true
6109 function Babel.math_box_dir(head)
6110 if Babel.use_math_box_dir then
6111 if not (token.get_macro('bbl@insidemath') == '0') then
6112 if Babel.hlist_has_bidi(head) then
6113 local d = node.new(node.id'dir')
6114 d.dir = '+TRT'
6115 node.insert_before(head, node.has_glyph(head), d)
6116 for item in node.traverse(head) do
6117 node.set_attribute(item,
6118 Babel.attr_dir, token.get_macro('bbl@thedir'))
6119 end
6120 end
6121 end
6122 end
6123 return head
6124 end
6125 luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6126 "Babel.math_box_dir", 0)
6127 }}%
6128 \fi

```



## 12.11 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.

```
6129 \bbl@trace{Redefinitions for bidi layout}
6130 %
6131 <<(*More package options)>> ≡
6132 \chardef\bbl@eqnpos\z@
6133 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6134 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6135 <</More package options>>
6136 %
6137 \def\BabelNoAMSMath{\let\bbl@noamsmath\relax}
6138 \ifnum\bbl@bidimode>\z@
6139   \ifx\matheqdirmode\@undefined\else
6140     \matheqdirmode\@ne
6141   \fi
6142   \let\bbl@eqnodir\relax
6143   \def\bbl@eqdel{()}
6144   \def\bbl@eqnum{%
6145     {\normalfont\normalcolor
6146       \expandafter\@firstoftwo\bbl@eqdel
6147       \theequation
6148       \expandafter\@secondoftwo\bbl@eqdel}}
6149   \def\bbl@puteqno#1{\eqno\hbox{#1}}
6150   \def\bbl@putleqno#1{\leqno\hbox{#1}}
6151   \def\bbl@eqno@flip#1{%
6152     \ifdim\predisplaysize=-\maxdimen
6153       \eqno
6154       \hb@xt@.01pt{\hb@xt@\displaywidth{\hss{#1}}\hss}%
6155     \else
6156       \leqno\hbox{#1}%
6157     \fi}
6158   \def\bbl@leqno@flip#1{%
6159     \ifdim\predisplaysize=-\maxdimen
6160       \leqno
6161       \hb@xt@.01pt{\hss\hb@xt@\displaywidth{{#1}\hss}}%
6162     \else
6163       \eqno\hbox{#1}%
6164     \fi}
6165   \AtBeginDocument{%
6166     \ifx\maketag@@@\@undefined % Normal equation, eqnarray
6167       \AddToHook{env/equation/begin}{%
6168         \ifnum\bbl@thetextdir>\z@
6169           \let\bbl@mathboxdir\relax
6170           \let\@eqnnum\bbl@eqnum
6171           \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6172           \chardef\bbl@thetextdir\z@
6173           \bbl@add\normalfont{\bbl@eqnodir}%
6174           \ifcase\bbl@eqnpos
6175             \let\bbl@puteqno\bbl@eqno@flip
6176           \or
6177             \let\bbl@puteqno\bbl@leqno@flip
```

```

6178         \fi
6179     \fi}%
6180 \ifnum\bb1@eqnpos=\tw@ \else
6181     \def\endequation{\bb1@puteqno{\@eqnnum}$$\@ignoretrue}%
6182 \fi
6183 \AddToHook{env/eqnarray/begin}{%
6184     \ifnum\bb1@thetextdir>\z@
6185         \let\bb1@mathboxdir\relax
6186         \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6187         \chardef\bb1@thetextdir\z@
6188         \bb1@add\normalfont{\bb1@eqnodir}%
6189         \ifnum\bb1@eqnpos=\@ne
6190             \def\@eqnnum{%
6191                 \setbox\z@\hbox{\bb1@eqnum}%
6192                 \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6193             \else
6194                 \let\@eqnnum\bb1@eqnum
6195             \fi
6196         \fi}
6197 % Hack. YA luatex bug?:
6198 \expandafter\bb1@sreplace\csname] \endcsname{$$}{\eqno\kern.001pt$$}%
6199 \else % amstex
6200     \ifx\bb1@noamsmath\undefined
6201         \bb1@exp{% Hack to hide maybe undefined conditionals:
6202             \chardef\bb1@eqnpos=0%
6203             \<iftagsleft@>1<else>\<if@fleqn>2<fi>\<fi>\relax}%
6204             \ifnum\bb1@eqnpos=\@ne
6205                 \let\bb1@ams@lap\hbox
6206             \else
6207                 \let\bb1@ams@lap\llap
6208             \fi
6209             \ExplSyntaxOn
6210             \bb1@sreplace\intertext@{\normalbaselines}%
6211             {\normalbaselines
6212                 \ifx\bb1@eqnodir\relax\else\bb1@pardir\@ne\bb1@eqnodir\fi}%
6213             \ExplSyntaxOff
6214             \def\bb1@ams@tagbox#1#2{#1{\bb1@eqnodir#2}}% #1=hbox|@lap|flip
6215             \ifx\bb1@ams@lap\hbox % leqno
6216                 \def\bb1@ams@flip#1{%
6217                     \hbox to 0.01pt{\hss\hbox to\displaywidth{\{#1}\hss}}}%
6218             \else % eqno
6219                 \def\bb1@ams@flip#1{%
6220                     \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}}\hss}}%
6221             \fi
6222             \def\bb1@ams@preset#1{%
6223                 \let\bb1@mathboxdir\relax
6224                 \ifnum\bb1@thetextdir>\z@
6225                     \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6226                     \bb1@sreplace\textdef@{\hbox}{\bb1@ams@tagbox\hbox}%
6227                     \bb1@sreplace\maketag@@@{\hbox}{\bb1@ams@tagbox#1}%
6228                 \fi}%
6229             \ifnum\bb1@eqnpos=\tw@ \else
6230                 \def\bb1@ams@equation{%
6231                     \let\bb1@mathboxdir\relax
6232                     \ifnum\bb1@thetextdir>\z@
6233                         \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6234                         \chardef\bb1@thetextdir\z@
6235                         \bb1@add\normalfont{\bb1@eqnodir}%
6236                         \ifcase\bb1@eqnpos
6237                             \def\veqno##1##2{\bb1@eqno@flip{##1##2}}%
6238                         \or
6239                             \def\veqno##1##2{\bb1@leqno@flip{##1##2}}%
6240                         \fi

```

```

6241         \fi}%
6242         \AddToHook{env/equation/begin}{\bbl@ams@equation}%
6243         \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6244     \fi
6245     \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6246     \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6247     \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6248     \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6249     \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6250     \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6251     \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6252     % Hackish, for proper alignment. Don't ask me why it works!:
6253     \bbl@exp{% Avoid a 'visible' conditional
6254         \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\tag*{}\<fi>}}%
6255     \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6256     \AddToHook{env/split/before}{%
6257         \let\bbl@mathboxdir\relax
6258         \ifnum\bbl@thetextdir>\z@
6259             \bbl@ifsamestring\@currentenv{equation}%
6260             {\ifx\bbl@ams@lap\hbox % leqno
6261                 \def\bbl@ams@flip#1{%
6262                     \hbox to 0.01pt{\hbox to\displaywidth{{#1}\hss}\hss}}%
6263                 \else
6264                     \def\bbl@ams@flip#1{%
6265                         \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}}%
6266                 \fi}%
6267             }%
6268         \fi}%
6269     \fi
6270 \fi}
6271 \fi
6272 \def\bbl@provide@extra#1{%
6273     % == Counters: mapdigits ==
6274     % Native digits
6275     \ifx\bbl@KVP@mapdigits\@nnil\else
6276         \bbl@ifunset{\bbl@dgnat@\language\name}{}%
6277         {\RequirePackage{luatexbase}%
6278         \bbl@activate@preotf
6279         \directlua{
6280             Babel = Babel or {} %%% -> presets in luababel
6281             Babel.digits_mapped = true
6282             Babel.digits = Babel.digits or {}
6283             Babel.digits[\the\localeid] =
6284                 table.pack(string.utfvalue('\bbl@cl{dgnat}'))
6285             if not Babel.numbers then
6286                 function Babel.numbers(head)
6287                     local LOCALE = Babel.attr_locale
6288                     local GLYPH = node.id'glyph'
6289                     local inmath = false
6290                     for item in node.traverse(head) do
6291                         if not inmath and item.id == GLYPH then
6292                             local temp = node.get_attribute(item, LOCALE)
6293                             if Babel.digits[temp] then
6294                                 local chr = item.char
6295                                 if chr > 47 and chr < 58 then
6296                                     item.char = Babel.digits[temp][chr-47]
6297                                 end
6298                             end
6299                         elseif item.id == node.id'math' then
6300                             inmath = (item.subtype == 0)
6301                         end
6302                     end
6303                     return head

```

```

6304         end
6305     end
6306 }}%
6307 \fi
6308 % == transforms ==
6309 \ifx\bb1@KVP@transforms\@nnil\else
6310     \def\bb1@elt##1##2##3{%
6311         \in@{$transforms.}{$##1}%
6312         \ifin@
6313             \def\bb1@tempa{##1}%
6314             \bb1@replace\bb1@tempa{transforms.}{}%
6315             \bb1@carg\bb1@transforms{babel\bb1@tempa}{##2}{##3}%
6316         \fi}%
6317     \csname bbl@inidata@\language\endcsname
6318     \bb1@release@transforms\relax % \relax closes the last item.
6319 \fi}
6320 \ifx\bb1@opt@layout\@nnil\endinput\fi % if no layout
6321 %
6322 \ifnum\bb1@bidimode>\z@
6323     \def\bb1@nextfake#1{% non-local changes, use always inside a group!
6324         \bb1@exp{%
6325             \def\bb1@insidemath{0}%
6326             \mathdir\the\bodydir
6327             #1%           Once entered in math, set boxes to restore values
6328             \<ifmmode>%
6329                 \everyvbox{%
6330                     \the\everyvbox
6331                     \bodydir\the\bodydir
6332                     \mathdir\the\mathdir
6333                     \everyhbox{\the\everyhbox}%
6334                     \everyvbox{\the\everyvbox}}%
6335                 \everyhbox{%
6336                     \the\everyhbox
6337                     \bodydir\the\bodydir
6338                     \mathdir\the\mathdir
6339                     \everyhbox{\the\everyhbox}%
6340                     \everyvbox{\the\everyvbox}}%
6341             \<fi>}}%
6342     \def\@hangfrom#1{%
6343         \setbox\@tempboxa\hbox{#1}%
6344         \hangindent\wd\@tempboxa
6345         \ifnum\bb1@getluadir{page}=\bb1@getluadir{par}\else
6346             \shapemode\@ne
6347         \fi
6348         \noindent\box\@tempboxa}
6349 \fi
6350 \IfBabelLayout{tabular}
6351 {\let\bb1@OL@tabular\@tabular
6352  \bb1@replace\@tabular{$}{\bb1@nextfake$}%
6353  \let\bb1@NL@tabular\@tabular
6354  \AtBeginDocument{%
6355      \ifx\bb1@NL@tabular\@tabular\else
6356          \bb1@replace\@tabular{$}{\bb1@nextfake$}%
6357          \let\bb1@NL@tabular\@tabular
6358      \fi}}
6359 {}
6360 \IfBabelLayout{lists}
6361 {\let\bb1@OL@list\list
6362  \bb1@sreplace\list{\parshape}{\bb1@listparshape}%
6363  \let\bb1@NL@list\list
6364  \def\bb1@listparshape#1#2#3{%
6365      \parshape #1 #2 #3 %
6366      \ifnum\bb1@getluadir{page}=\bb1@getluadir{par}\else

```

```

6367     \shapemode\tw@
6368     \fi}}
6369 {}
6370 \IfBabelLayout{graphics}
6371 {\let\bbbl@pictresetdir\relax
6372  \def\bbbl@pictsetdir#1{%
6373   \ifcase\bbbl@thetextdir
6374   \let\bbbl@pictresetdir\relax
6375   \else
6376   \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6377   \or\textdir TLT
6378   \else\bodydir TLT \textdir TLT
6379   \fi
6380   % \{(text|par)dir required in pgf:
6381   \def\bbbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6382   \fi}%
6383 \AddToHook{env/picture/begin}{\bbbl@pictsetdir\tw@}%
6384 \directlua{
6385   Babel.get_picture_dir = true
6386   Babel.picture_has_bidi = 0
6387   %
6388   function Babel.picture_dir (head)
6389     if not Babel.get_picture_dir then return head end
6390     if Babel.hlist_has_bidi(head) then
6391       Babel.picture_has_bidi = 1
6392     end
6393     return head
6394   end
6395   luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6396   "Babel.picture_dir")
6397 }%
6398 \AtBeginDocument{%
6399   \def\LS@rot{%
6400     \setbox\@outputbox\ vbox{%
6401       \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
6402   \long\def\put(#1,#2)#3{%
6403     \@killglue
6404     % Try:
6405     \ifx\bbbl@pictresetdir\relax
6406       \def\bbbl@tempc{0}%
6407     \else
6408       \directlua{
6409         Babel.get_picture_dir = true
6410         Babel.picture_has_bidi = 0
6411       }%
6412       \setbox\z@\hb@xt@\z@{%
6413         \@defaultunitsset\@tempdimc{#1}\unitlength
6414         \kern\@tempdimc
6415         #3\hss}% TODO: #3 executed twice (below). That's bad.
6416       \edef\bbbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6417       \fi
6418       % Do:
6419       \@defaultunitsset\@tempdimc{#2}\unitlength
6420       \raise\@tempdimc\hb@xt@\z@{%
6421         \@defaultunitsset\@tempdimc{#1}\unitlength
6422         \kern\@tempdimc
6423         {\ifnum\bbbl@tempc>\z@\bbbl@pictresetdir\fi#3}\hss}%
6424       \ignorespaces}%
6425   \MakeRobust\put}%
6426 \AtBeginDocument
6427 {\AddToHook{cmd/diagbox@pict/before}{\let\bbbl@pictsetdir\@gobble}%
6428  \ifx\pgfpicture\undefined\else % TODO. Allow deactivate?
6429  \AddToHook{env/pgfpicture/begin}{\bbbl@pictsetdir\@ne}%

```

```

6430 \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6431 \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6432 \fi
6433 \ifx\tikzpicture\undefined\else
6434 \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\z@}%
6435 \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6436 \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
6437 \fi
6438 \ifx\tcolorbox\undefined\else
6439 \def\tcb@drawing@env@begin{%
6440 \csname tcb@before@tcb@split@state\endcsname
6441 \bbl@pictsetdir\tw@
6442 \begin{\kvtcb@graphenv}%
6443 \tcb@bbdraw%
6444 \tcb@apply@graph@patches
6445 }%
6446 \def\tcb@drawing@env@end{%
6447 \end{\kvtcb@graphenv}%
6448 \bbl@pictresetdir
6449 \csname tcb@after@tcb@split@state\endcsname
6450 }%
6451 \fi
6452 }}
6453 {}

```

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`.

```

6454 \IfBabelLayout{counters*}%
6455 {\bbl@add\bbl@opt@layout{.counters.}%
6456 \directlua{
6457 \luaexec{
6458 \luaexec{
6459 }}}}
6460 \IfBabelLayout{counters}%
6461 {\let\bbl@OL@@textsuperscript\textsuperscript
6462 \bbl@sreplace\textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6463 \let\bbl@Latinarabic=\@arabic
6464 \let\bbl@OL@@arabic\@arabic
6465 \def\@arabic#1{\babelsublr{\bbl@Latinarabic#1}}%
6466 \ifpackagewith{babel}{bidi=default}%
6467 {\let\bbl@asciroman=\@roman
6468 \let\bbl@OL@@roman\@roman
6469 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
6470 \let\bbl@asciRoman=\@Roman
6471 \let\bbl@OL@@roman\@Roman
6472 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciRoman#1}}}%
6473 \let\bbl@OL@labelenumii\labelenumii
6474 \def\labelenumii{\theenumii}%
6475 \let\bbl@OL@p@enumiii\p@enumiii
6476 \def\p@enumiii{\p@enumii}\theenumii}}}}
6477 <<Footnote changes>>
6478 \IfBabelLayout{footnotes}%
6479 {\let\bbl@OL@footnote\footnote
6480 \BabelFootnote\footnote\language\{}}%
6481 \BabelFootnote\localfootnote\language\{}}%
6482 \BabelFootnote\mainfootnote\{}}%
6483 {}

```

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.

```

6484 \IfBabelLayout{extras}%
6485 {\let\bbl@OL@underline\underline
6486 \bbl@sreplace\underline{\$@@underline}{\bbl@nextfake$@@underline}%

```

```

6487 \let\bbl@OL@LaTeX2e\LaTeX2e
6488 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6489 \if b\expandafter\car\@series\@nil\boldmath\fi
6490 \babelsublr{%
6491 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}
6492 {}
6493 </luatex>

```

## 12.12 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.

```

6494 <*transforms>
6495 Babel.linebreaking.replacements = {}
6496 Babel.linebreaking.replacements[0] = {} -- pre
6497 Babel.linebreaking.replacements[1] = {} -- post
6498 Babel.linebreaking.replacements[2] = {} -- post-line WIP
6499
6500 -- Discretionaries contain strings as nodes
6501 function Babel.str_to_nodes(fn, matches, base)
6502   local n, head, last
6503   if fn == nil then return nil end
6504   for s in string.utfvalues(fn(matches)) do
6505     if base.id == 7 then
6506       base = base.replace
6507     end
6508     n = node.copy(base)
6509     n.char = s
6510     if not head then
6511       head = n
6512     else
6513       last.next = n
6514     end
6515     last = n
6516   end
6517   return head
6518 end
6519
6520 Babel.fetch_subtext = {}
6521
6522 Babel.ignore_pre_char = function(node)
6523   return (node.lang == Babel.nohyphenation)
6524 end
6525
6526 -- Merging both functions doesn't seem feasible, because there are too
6527 -- many differences.
6528 Babel.fetch_subtext[0] = function(head)
6529   local word_string = ''
6530   local word_nodes = {}
6531   local lang
6532   local item = head
6533   local inmath = false
6534
6535   while item do

```

```

6536
6537     if item.id == 11 then
6538         inmath = (item.subtype == 0)
6539     end
6540
6541     if inmath then
6542         -- pass
6543
6544     elseif item.id == 29 then
6545         local locale = node.get_attribute(item, Babel.attr_locale)
6546
6547         if lang == locale or lang == nil then
6548             lang = lang or locale
6549             if Babel.ignore_pre_char(item) then
6550                 word_string = word_string .. Babel.us_char
6551             else
6552                 word_string = word_string .. unicode.utf8.char(item.char)
6553             end
6554             word_nodes[#word_nodes+1] = item
6555         else
6556             break
6557         end
6558
6559     elseif item.id == 12 and item.subtype == 13 then
6560         word_string = word_string .. ' '
6561         word_nodes[#word_nodes+1] = item
6562
6563         -- Ignore leading unrecognized nodes, too.
6564     elseif word_string ~= '' then
6565         word_string = word_string .. Babel.us_char
6566         word_nodes[#word_nodes+1] = item -- Will be ignored
6567     end
6568
6569     item = item.next
6570 end
6571
6572 -- Here and above we remove some trailing chars but not the
6573 -- corresponding nodes. But they aren't accessed.
6574 if word_string:sub(-1) == ' ' then
6575     word_string = word_string:sub(1,-2)
6576 end
6577 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6578 return word_string, word_nodes, item, lang
6579 end
6580
6581 Babel.fetch_subtext[1] = function(head)
6582     local word_string = ''
6583     local word_nodes = {}
6584     local lang
6585     local item = head
6586     local inmath = false
6587
6588     while item do
6589
6590         if item.id == 11 then
6591             inmath = (item.subtype == 0)
6592         end
6593
6594         if inmath then
6595             -- pass
6596
6597         elseif item.id == 29 then
6598             if item.lang == lang or lang == nil then

```



```

6599         if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6600             lang = lang or item.lang
6601             word_string = word_string .. unicode.utf8.char(item.char)
6602             word_nodes[#word_nodes+1] = item
6603         end
6604     else
6605         break
6606     end
6607
6608 elseif item.id == 7 and item.subtype == 2 then
6609     word_string = word_string .. '='
6610     word_nodes[#word_nodes+1] = item
6611
6612 elseif item.id == 7 and item.subtype == 3 then
6613     word_string = word_string .. '|'
6614     word_nodes[#word_nodes+1] = item
6615
6616 -- (1) Go to next word if nothing was found, and (2) implicitly
6617 -- remove leading USs.
6618 elseif word_string == '' then
6619     -- pass
6620
6621 -- This is the responsible for splitting by words.
6622 elseif (item.id == 12 and item.subtype == 13) then
6623     break
6624
6625 else
6626     word_string = word_string .. Babel.us_char
6627     word_nodes[#word_nodes+1] = item -- Will be ignored
6628 end
6629
6630 item = item.next
6631 end
6632
6633 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6634 return word_string, word_nodes, item, lang
6635 end
6636
6637 function Babel.pre_hyphenate_replace(head)
6638     Babel.hyphenate_replace(head, 0)
6639 end
6640
6641 function Babel.post_hyphenate_replace(head)
6642     Babel.hyphenate_replace(head, 1)
6643 end
6644
6645 Babel.us_char = string.char(31)
6646
6647 function Babel.hyphenate_replace(head, mode)
6648     local u = unicode.utf8
6649     local lbkr = Babel.linebreaking.replacements[mode]
6650     if mode == 2 then mode = 0 end -- WIP
6651
6652     local word_head = head
6653
6654     while true do -- for each subtext block
6655
6656         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6657
6658         if Babel.debug then
6659             print()
6660             print((mode == 0) and '@@@<' or '@@@>', w)
6661         end

```

```

6662
6663     if nw == nil and w == '' then break end
6664
6665     if not lang then goto next end
6666     if not lbkr[lang] then goto next end
6667
6668     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6669     -- loops are nested.
6670     for k=1, #lbkr[lang] do
6671         local p = lbkr[lang][k].pattern
6672         local r = lbkr[lang][k].replace
6673         local attr = lbkr[lang][k].attr or -1
6674
6675         if Babel.debug then
6676             print('*****', p, mode)
6677         end
6678
6679         -- This variable is set in some cases below to the first *byte*
6680         -- after the match, either as found by u.match (faster) or the
6681         -- computed position based on sc if w has changed.
6682         local last_match = 0
6683         local step = 0
6684
6685         -- For every match.
6686         while true do
6687             if Babel.debug then
6688                 print('====')
6689             end
6690             local new -- used when inserting and removing nodes
6691
6692             local matches = { u.match(w, p, last_match) }
6693
6694             if #matches < 2 then break end
6695
6696             -- Get and remove empty captures (with ()'s, which return a
6697             -- number with the position), and keep actual captures
6698             -- (from (...)), if any, in matches.
6699             local first = table.remove(matches, 1)
6700             local last = table.remove(matches, #matches)
6701             -- Non re-fetched substrings may contain \31, which separates
6702             -- substrings.
6703             if string.find(w:sub(first, last-1), Babel.us_char) then break end
6704
6705             local save_last = last -- with A()BC()D, points to D
6706
6707             -- Fix offsets, from bytes to unicode. Explained above.
6708             first = u.len(w:sub(1, first-1)) + 1
6709             last = u.len(w:sub(1, last-1)) -- now last points to C
6710
6711             -- This loop stores in a small table the nodes
6712             -- corresponding to the pattern. Used by 'data' to provide a
6713             -- predictable behavior with 'insert' (w_nodes is modified on
6714             -- the fly), and also access to 'remove'd nodes.
6715             local sc = first-1 -- Used below, too
6716             local data_nodes = {}
6717
6718             local enabled = true
6719             for q = 1, last-first+1 do
6720                 data_nodes[q] = w_nodes[sc+q]
6721                 if enabled
6722                     and attr > -1
6723                     and not node.has_attribute(data_nodes[q], attr)
6724                 then

```

```

6725         enabled = false
6726     end
6727 end
6728
6729 -- This loop traverses the matched substring and takes the
6730 -- corresponding action stored in the replacement list.
6731 -- sc = the position in substr nodes / string
6732 -- rc = the replacement table index
6733 local rc = 0
6734
6735 while rc < last-first+1 do -- for each replacement
6736     if Babel.debug then
6737         print('.....', rc + 1)
6738     end
6739     sc = sc + 1
6740     rc = rc + 1
6741
6742     if Babel.debug then
6743         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6744         local ss = ''
6745         for itt in node.traverse(head) do
6746             if itt.id == 29 then
6747                 ss = ss .. unicode.utf8.char(itt.char)
6748             else
6749                 ss = ss .. '{' .. itt.id .. '}'
6750             end
6751         end
6752         print('*****', ss)
6753     end
6754
6755     local crep = r[rc]
6756     local item = w_nodes[sc]
6757     local item_base = item
6758     local placeholder = Babel.us_char
6759     local d
6760
6761     if crep and crep.data then
6762         item_base = data_nodes[crep.data]
6763     end
6764
6765     if crep then
6766         step = crep.step or 0
6767     end
6768
6769     if (not enabled) or (crep and next(crep) == nil) then -- = {}
6770         last_match = save_last -- Optimization
6771         goto next
6772     end
6773
6774     elseif crep == nil or crep.remove then
6775         node.remove(head, item)
6776         table.remove(w_nodes, sc)
6777         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6778         sc = sc - 1 -- Nothing has been inserted.
6779         last_match = utf8.offset(w, sc+1+step)
6780         goto next
6781
6782     elseif crep and crep.kashida then -- Experimental
6783         node.set_attribute(item,
6784             Babel.attr_kashida,
6785             crep.kashida)
6786         last_match = utf8.offset(w, sc+1+step)
6787         goto next

```

```

6788
6789 elseif crep and crep.string then
6790   local str = crep.string(matches)
6791   if str == '' then -- Gather with nil
6792     node.remove(head, item)
6793     table.remove(w_nodes, sc)
6794     w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6795     sc = sc - 1 -- Nothing has been inserted.
6796   else
6797     local loop_first = true
6798     for s in string.utfvalues(str) do
6799       d = node.copy(item_base)
6800       d.char = s
6801       if loop_first then
6802         loop_first = false
6803         head, new = node.insert_before(head, item, d)
6804         if sc == 1 then
6805           word_head = head
6806         end
6807         w_nodes[sc] = d
6808         w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6809       else
6810         sc = sc + 1
6811         head, new = node.insert_before(head, item, d)
6812         table.insert(w_nodes, sc, new)
6813         w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6814       end
6815       if Babel.debug then
6816         print('.....', 'str')
6817         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6818       end
6819     end -- for
6820     node.remove(head, item)
6821   end -- if ''
6822   last_match = utf8.offset(w, sc+1+step)
6823   goto next
6824
6825 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6826   d = node.new(7, 0) -- (disc, discretionary)
6827   d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
6828   d.post = Babel.str_to_nodes(crep.post, matches, item_base)
6829   d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6830   d.attr = item_base.attr
6831   if crep.pre == nil then -- TeXbook p96
6832     d.penalty = crep.penalty or tex.hyphenpenalty
6833   else
6834     d.penalty = crep.penalty or tex.exhyphenpenalty
6835   end
6836   placeholder = '|'
6837   head, new = node.insert_before(head, item, d)
6838
6839 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6840   -- ERROR
6841
6842 elseif crep and crep.penalty then
6843   d = node.new(14, 0) -- (penalty, userpenalty)
6844   d.attr = item_base.attr
6845   d.penalty = crep.penalty
6846   head, new = node.insert_before(head, item, d)
6847
6848 elseif crep and crep.space then
6849   -- 655360 = 10 pt = 10 * 65536 sp
6850   d = node.new(12, 13) -- (glue, spaceskip)

```

```

6851         local quad = font.getfont(item_base.font).size or 655360
6852         node.setglue(d, crep.space[1] * quad,
6853                     crep.space[2] * quad,
6854                     crep.space[3] * quad)
6855         if mode == 0 then
6856             placeholder = ' '
6857         end
6858         head, new = node.insert_before(head, item, d)
6859
6860     elseif crep and crep.spacefactor then
6861         d = node.new(12, 13) -- (glue, spaceskip)
6862         local base_font = font.getfont(item_base.font)
6863         node.setglue(d,
6864                     crep.spacefactor[1] * base_font.parameters['space'],
6865                     crep.spacefactor[2] * base_font.parameters['space_stretch'],
6866                     crep.spacefactor[3] * base_font.parameters['space_shrink'])
6867         if mode == 0 then
6868             placeholder = ' '
6869         end
6870         head, new = node.insert_before(head, item, d)
6871
6872     elseif mode == 0 and crep and crep.space then
6873         -- ERROR
6874
6875     end -- ie replacement cases
6876
6877     -- Shared by disc, space and penalty.
6878     if sc == 1 then
6879         word_head = head
6880     end
6881     if crep.insert then
6882         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6883         table.insert(w_nodes, sc, new)
6884         last = last + 1
6885     else
6886         w_nodes[sc] = d
6887         node.remove(head, item)
6888         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6889     end
6890
6891     last_match = utf8.offset(w, sc+1+step)
6892
6893     ::next::
6894
6895     end -- for each replacement
6896
6897     if Babel.debug then
6898         print('.....', '/')
6899         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6900     end
6901
6902     end -- for match
6903
6904     end -- for patterns
6905
6906     ::next::
6907     word_head = nw
6908     end -- for substring
6909     return head
6910 end
6911
6912 -- This table stores capture maps, numbered consecutively
6913 Babel.capture_maps = {}

```

```

6914
6915 -- The following functions belong to the next macro
6916 function Babel.capture_func(key, cap)
6917   local ret = "[" .. cap:gsub('{{[0-9]}}', "]]..m[%1]..[" .. "]"
6918   local cnt
6919   local u = unicode.utf8
6920   ret, cnt = ret:gsub('{{[0-9]}|^|+|(.-)}', Babel.capture_func_map)
6921   if cnt == 0 then
6922     ret = u.gsub(ret, '{{(%x%x%x%x+)}',
6923                 function (n)
6924                   return u.char(tonumber(n, 16))
6925                 end)
6926   end
6927   ret = ret:gsub("%[%]%%.", '')
6928   ret = ret:gsub("%.%[%]%%", '')
6929   return key .. "[[=function(m) return ]] .. ret .. [[ end]]
6930 end
6931
6932 function Babel.capt_map(from, mapno)
6933   return Babel.capture_maps[mapno][from] or from
6934 end
6935
6936 -- Handle the {n|abc|ABC} syntax in captures
6937 function Babel.capture_func_map(capno, from, to)
6938   local u = unicode.utf8
6939   from = u.gsub(from, '{{(%x%x%x%x+)}',
6940               function (n)
6941                 return u.char(tonumber(n, 16))
6942               end)
6943   to = u.gsub(to, '{{(%x%x%x%x+)}',
6944             function (n)
6945               return u.char(tonumber(n, 16))
6946             end)
6947   local froms = {}
6948   for s in string.utfcharacters(from) do
6949     table.insert(froms, s)
6950   end
6951   local cnt = 1
6952   table.insert(Babel.capture_maps, {})
6953   local mlen = table.getn(Babel.capture_maps)
6954   for s in string.utfcharacters(to) do
6955     Babel.capture_maps[mlen][froms[cnt]] = s
6956     cnt = cnt + 1
6957   end
6958   return "]]..Babel.capt_map(m[" .. capno .. "], " ..
6959         (mlen) .. ").." .. "["
6960 end
6961
6962 -- Create/Extend reversed sorted list of kashida weights:
6963 function Babel.capture_kashida(key, wt)
6964   wt = tonumber(wt)
6965   if Babel.kashida_wts then
6966     for p, q in ipairs(Babel.kashida_wts) do
6967       if wt == q then
6968         break
6969       elseif wt > q then
6970         table.insert(Babel.kashida_wts, p, wt)
6971         break
6972       elseif table.getn(Babel.kashida_wts) == p then
6973         table.insert(Babel.kashida_wts, wt)
6974       end
6975     end
6976   else

```

```

6977     Babel.kashida_wts = { wt }
6978   end
6979   return 'kashida = ' .. wt
6980 end
6981 </transforms>

```

## 12.13 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.

```

6982 <*basic-r>
6983 Babel = Babel or {}
6984
6985 Babel.bidi_enabled = true
6986
6987 require('babel-data-bidi.lua')
6988
6989 local characters = Babel.characters
6990 local ranges = Babel.ranges
6991
6992 local DIR = node.id("dir")
6993
6994 local function dir_mark(head, from, to, outer)
6995   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6996   local d = node.new(DIR)
6997   d.dir = '+' .. dir
6998   node.insert_before(head, from, d)
6999   d = node.new(DIR)
7000   d.dir = '-' .. dir

```

```

7001 node.insert_after(head, to, d)
7002 end
7003
7004 function Babel.bidi(head, ispar)
7005   local first_n, last_n          -- first and last char with nums
7006   local last_es                  -- an auxiliary 'last' used with nums
7007   local first_d, last_d          -- first and last char in L/R block
7008   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):

```

7009   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7010   local strong_lr = (strong == 'l') and 'l' or 'r'
7011   local outer = strong
7012
7013   local new_dir = false
7014   local first_dir = false
7015   local inmath = false
7016
7017   local last_lr
7018
7019   local type_n = ''
7020
7021   for item in node.traverse(head) do
7022
7023     -- three cases: glyph, dir, otherwise
7024     if item.id == node.id'glyph'
7025       or (item.id == 7 and item.subtype == 2) then
7026
7027       local itemchar
7028       if item.id == 7 and item.subtype == 2 then
7029         itemchar = item.replace.char
7030       else
7031         itemchar = item.char
7032       end
7033       local chardata = characters[itemchar]
7034       dir = chardata and chardata.d or nil
7035       if not dir then
7036         for nn, et in ipairs(ranges) do
7037           if itemchar < et[1] then
7038             break
7039           elseif itemchar <= et[2] then
7040             dir = et[3]
7041             break
7042           end
7043         end
7044       end
7045       dir = dir or 'l'
7046       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).

```

7047   if new_dir then
7048     attr_dir = 0
7049     for at in node.traverse(item.attr) do
7050       if at.number == Babel.attr_dir then
7051         attr_dir = at.value % 3
7052       end
7053     end
7054     if attr_dir == 1 then

```



```

7055         strong = 'r'
7056     elseif attr_dir == 2 then
7057         strong = 'al'
7058     else
7059         strong = 'l'
7060     end
7061     strong_lr = (strong == 'l') and 'l' or 'r'
7062     outer = strong_lr
7063     new_dir = false
7064 end
7065
7066     if dir == 'nsm' then dir = strong end          -- W1

```

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

```

7067     dir_real = dir          -- We need dir_real to set strong below
7068     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:

```

7069     if strong == 'al' then
7070         if dir == 'en' then dir = 'an' end          -- W2
7071         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7072         strong_lr = 'r'          -- W3
7073     end

```

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

```

7074     elseif item.id == node.id'dir' and not inmath then
7075         new_dir = true
7076         dir = nil
7077     elseif item.id == node.id'math' then
7078         inmath = (item.subtype == 0)
7079     else
7080         dir = nil          -- Not a char
7081     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>.

```

7082     if dir == 'en' or dir == 'an' or dir == 'et' then
7083         if dir ~= 'et' then
7084             type_n = dir
7085         end
7086         first_n = first_n or item
7087         last_n = last_es or item
7088         last_es = nil
7089     elseif dir == 'es' and last_n then -- W3+W6
7090         last_es = item
7091     elseif dir == 'cs' then          -- it's right - do nothing
7092     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7093         if strong_lr == 'r' and type_n ~= '' then
7094             dir_mark(head, first_n, last_n, 'r')
7095         elseif strong_lr == 'l' and first_d and type_n == 'an' then
7096             dir_mark(head, first_n, last_n, 'r')
7097             dir_mark(head, first_d, last_d, outer)
7098             first_d, last_d = nil, nil
7099         elseif strong_lr == 'l' and type_n ~= '' then
7100             last_d = last_n
7101         end
7102         type_n = ''
7103         first_n, last_n = nil, nil
7104     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:

```

7105   if dir == 'l' or dir == 'r' then
7106       if dir ~= outer then
7107           first_d = first_d or item
7108           last_d = item
7109       elseif first_d and dir ~= strong_lr then
7110           dir_mark(head, first_d, last_d, outer)
7111           first_d, last_d = nil, nil
7112       end
7113   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.

```

7114   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7115       item.char = characters[item.char] and
7116           characters[item.char].m or item.char
7117   elseif (dir or new_dir) and last_lr ~= item then
7118       local mir = outer .. strong_lr .. (dir or outer)
7119       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7120           for ch in node.traverse(node.next(last_lr)) do
7121               if ch == item then break end
7122               if ch.id == node.id'glyph' and characters[ch.char] then
7123                   ch.char = characters[ch.char].m or ch.char
7124               end
7125           end
7126       end
7127   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).

```

7128   if dir == 'l' or dir == 'r' then
7129       last_lr = item
7130       strong = dir_real           -- Don't search back - best save now
7131       strong_lr = (strong == 'l') and 'l' or 'r'
7132   elseif new_dir then
7133       last_lr = nil
7134   end
7135 end

```

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

```

7136   if last_lr and outer == 'r' then
7137       for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7138           if characters[ch.char] then
7139               ch.char = characters[ch.char].m or ch.char
7140           end
7141       end
7142   end
7143   if first_n then
7144       dir_mark(head, first_n, last_n, outer)
7145   end
7146   if first_d then
7147       dir_mark(head, first_d, last_d, outer)
7148   end

```

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

```

7149   return node.prev(head) or head

```

```

7150 end
7151 </basic-r>

And here the Lua code for bidi=basic:

7152 <*basic>
7153 Babel = Babel or {}
7154
7155 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
7156
7157 Babel.fontmap = Babel.fontmap or {}
7158 Babel.fontmap[0] = {}      -- l
7159 Babel.fontmap[1] = {}      -- r
7160 Babel.fontmap[2] = {}      -- al/an
7161
7162 Babel.bidi_enabled = true
7163 Babel.mirroring_enabled = true
7164
7165 require('babel-data-bidi.lua')
7166
7167 local characters = Babel.characters
7168 local ranges = Babel.ranges
7169
7170 local DIR = node.id('dir')
7171 local GLYPH = node.id('glyph')
7172
7173 local function insert_implicit(head, state, outer)
7174   local new_state = state
7175   if state.sim and state.eim and state.sim ~= state.eim then
7176     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
7177     local d = node.new(DIR)
7178     d.dir = '+' .. dir
7179     node.insert_before(head, state.sim, d)
7180     local d = node.new(DIR)
7181     d.dir = '-' .. dir
7182     node.insert_after(head, state.eim, d)
7183   end
7184   new_state.sim, new_state.eim = nil, nil
7185   return head, new_state
7186 end
7187
7188 local function insert_numeric(head, state)
7189   local new
7190   local new_state = state
7191   if state.san and state.ean and state.san ~= state.ean then
7192     local d = node.new(DIR)
7193     d.dir = '+TLT'
7194     _, new = node.insert_before(head, state.san, d)
7195     if state.san == state.sim then state.sim = new end
7196     local d = node.new(DIR)
7197     d.dir = '-TLT'
7198     _, new = node.insert_after(head, state.ean, d)
7199     if state.ean == state.eim then state.eim = new end
7200   end
7201   new_state.san, new_state.ean = nil, nil
7202   return head, new_state
7203 end
7204
7205 -- TODO - \hbox with an explicit dir can lead to wrong results
7206 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7207 -- was s made to improve the situation, but the problem is the 3-dir
7208 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7209 -- well.
7210

```

```

7211 function Babel.bidi(head, ispar, hdir)
7212   local d    -- d is used mainly for computations in a loop
7213   local prev_d = ''
7214   local new_d = false
7215
7216   local nodes = {}
7217   local outer_first = nil
7218   local inmath = false
7219
7220   local glue_d = nil
7221   local glue_i = nil
7222
7223   local has_en = false
7224   local first_et = nil
7225
7226   local has_hyperlink = false
7227
7228   local ATDIR = Babel.attr_dir
7229
7230   local save_outer
7231   local temp = node.get_attribute(head, ATDIR)
7232   if temp then
7233     temp = temp % 3
7234     save_outer = (temp == 0 and 'l') or
7235                  (temp == 1 and 'r') or
7236                  (temp == 2 and 'al')
7237   elseif ispar then -- Or error? Shouldn't happen
7238     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7239   else -- Or error? Shouldn't happen
7240     save_outer = ('TRT' == hdir) and 'r' or 'l'
7241   end
7242   -- when the callback is called, we are just _after_ the box,
7243   -- and the textdir is that of the surrounding text
7244   -- if not ispar and hdir ~= tex.textdir then
7245   --   save_outer = ('TRT' == hdir) and 'r' or 'l'
7246   -- end
7247   local outer = save_outer
7248   local last = outer
7249   -- 'al' is only taken into account in the first, current loop
7250   if save_outer == 'al' then save_outer = 'r' end
7251
7252   local fontmap = Babel.fontmap
7253
7254   for item in node.traverse(head) do
7255
7256     -- In what follows, #node is the last (previous) node, because the
7257     -- current one is not added until we start processing the neutrals.
7258
7259     -- three cases: glyph, dir, otherwise
7260     if item.id == GLYPH
7261       or (item.id == 7 and item.subtype == 2) then
7262
7263       local d_font = nil
7264       local item_r
7265       if item.id == 7 and item.subtype == 2 then
7266         item_r = item.replace -- automatic discs have just 1 glyph
7267       else
7268         item_r = item
7269       end
7270       local chardata = characters[item_r.char]
7271       d = chardata and chardata.d or nil
7272       if not d or d == 'nsm' then
7273         for nn, et in ipairs(ranges) do

```

```

7274         if item_r.char < et[1] then
7275             break
7276         elseif item_r.char <= et[2] then
7277             if not d then d = et[3]
7278             elseif d == 'nsm' then d_font = et[3]
7279             end
7280             break
7281         end
7282     end
7283 end
7284 d = d or 'l'
7285
7286 -- A short 'pause' in bidi for mapfont
7287 d_font = d_font or d
7288 d_font = (d_font == 'l' and 0) or
7289           (d_font == 'nsm' and 0) or
7290           (d_font == 'r' and 1) or
7291           (d_font == 'al' and 2) or
7292           (d_font == 'an' and 2) or nil
7293 if d_font and fontmap and fontmap[d_font][item_r.font] then
7294     item_r.font = fontmap[d_font][item_r.font]
7295 end
7296
7297 if new_d then
7298     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7299     if inmath then
7300         attr_d = 0
7301     else
7302         attr_d = node.get_attribute(item, ATDIR)
7303         attr_d = attr_d % 3
7304     end
7305     if attr_d == 1 then
7306         outer_first = 'r'
7307         last = 'r'
7308     elseif attr_d == 2 then
7309         outer_first = 'r'
7310         last = 'al'
7311     else
7312         outer_first = 'l'
7313         last = 'l'
7314     end
7315     outer = last
7316     has_en = false
7317     first_et = nil
7318     new_d = false
7319 end
7320
7321 if glue_d then
7322     if (d == 'l' and 'l' or 'r') ~= glue_d then
7323         table.insert(nodes, {glue_i, 'on', nil})
7324     end
7325     glue_d = nil
7326     glue_i = nil
7327 end
7328
7329 elseif item.id == DIR then
7330     d = nil
7331     -- new_d = true
7332     if head ~= item then new_d = true end
7333
7334 elseif item.id == node.id'glue' and item.subtype == 13 then
7335     glue_d = d
7336     glue_i = item

```

```

7337     d = nil
7338
7339     elseif item.id == node.id'math' then
7340         inmath = (item.subtype == 0)
7341
7342     elseif item.id == 8 and item.subtype == 19 then
7343         has_hyperlink = true
7344
7345     else
7346         d = nil
7347     end
7348
7349     -- AL <= EN/ET/ES      -- W2 + W3 + W6
7350     if last == 'al' and d == 'en' then
7351         d = 'an'          -- W3
7352     elseif last == 'al' and (d == 'et' or d == 'es') then
7353         d = 'on'          -- W6
7354     end
7355
7356     -- EN + CS/ES + EN      -- W4
7357     if d == 'en' and #nodes >= 2 then
7358         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7359             and nodes[#nodes-1][2] == 'en' then
7360             nodes[#nodes][2] = 'en'
7361         end
7362     end
7363
7364     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
7365     if d == 'an' and #nodes >= 2 then
7366         if (nodes[#nodes][2] == 'cs')
7367             and nodes[#nodes-1][2] == 'an' then
7368             nodes[#nodes][2] = 'an'
7369         end
7370     end
7371
7372     -- ET/EN                  -- W5 + W7->l / W6->on
7373     if d == 'et' then
7374         first_et = first_et or (#nodes + 1)
7375     elseif d == 'en' then
7376         has_en = true
7377         first_et = first_et or (#nodes + 1)
7378     elseif first_et then      -- d may be nil here !
7379         if has_en then
7380             if last == 'l' then
7381                 temp = 'l'    -- W7
7382             else
7383                 temp = 'en'   -- W5
7384             end
7385         else
7386             temp = 'on'       -- W6
7387         end
7388         for e = first_et, #nodes do
7389             if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7390         end
7391         first_et = nil
7392         has_en = false
7393     end
7394
7395     -- Force mathdir in math if ON (currently works as expected only
7396     -- with 'l')
7397     if inmath and d == 'on' then
7398         d = ('TRT' == tex.mathdir) and 'r' or 'l'
7399     end

```

```

7400
7401   if d then
7402       if d == 'al' then
7403           d = 'r'
7404           last = 'al'
7405       elseif d == 'l' or d == 'r' then
7406           last = d
7407       end
7408       prev_d = d
7409       table.insert(nodes, {item, d, outer_first})
7410   end
7411
7412   outer_first = nil
7413
7414 end
7415
7416 -- TODO -- repeated here in case EN/ET is the last node. Find a
7417 -- better way of doing things:
7418 if first_et then      -- dir may be nil here !
7419     if has_en then
7420         if last == 'l' then
7421             temp = 'l'    -- W7
7422         else
7423             temp = 'en'   -- W5
7424         end
7425     else
7426         temp = 'on'      -- W6
7427     end
7428     for e = first_et, #nodes do
7429         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7430     end
7431 end
7432
7433 -- dummy node, to close things
7434 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7435
7436 ----- NEUTRAL -----
7437
7438 outer = save_outer
7439 last = outer
7440
7441 local first_on = nil
7442
7443 for q = 1, #nodes do
7444     local item
7445
7446     local outer_first = nodes[q][3]
7447     outer = outer_first or outer
7448     last = outer_first or last
7449
7450     local d = nodes[q][2]
7451     if d == 'an' or d == 'en' then d = 'r' end
7452     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7453
7454     if d == 'on' then
7455         first_on = first_on or q
7456     elseif first_on then
7457         if last == d then
7458             temp = d
7459         else
7460             temp = outer
7461         end
7462         for r = first_on, q - 1 do

```

```

7463     nodes[r][2] = temp
7464     item = nodes[r][1]    -- MIRRORING
7465     if Babel.mirroring_enabled and item.id == GLYPH
7466         and temp == 'r' and characters[item.char] then
7467         local font_mode = ''
7468         if item.font > 0 and font.fonts[item.font].properties then
7469             font_mode = font.fonts[item.font].properties.mode
7470         end
7471         if font_mode ~= 'harf' and font_mode ~= 'plug' then
7472             item.char = characters[item.char].m or item.char
7473         end
7474     end
7475 end
7476 first_on = nil
7477 end
7478
7479 if d == 'r' or d == 'l' then last = d end
7480 end
7481
7482 ----- IMPLICIT, REORDER -----
7483
7484 outer = save_outer
7485 last = outer
7486
7487 local state = {}
7488 state.has_r = false
7489
7490 for q = 1, #nodes do
7491
7492     local item = nodes[q][1]
7493
7494     outer = nodes[q][3] or outer
7495
7496     local d = nodes[q][2]
7497
7498     if d == 'nsm' then d = last end          -- W1
7499     if d == 'en' then d = 'an' end
7500     local isdir = (d == 'r' or d == 'l')
7501
7502     if outer == 'l' and d == 'an' then
7503         state.san = state.san or item
7504         state.ean = item
7505     elseif state.san then
7506         head, state = insert_numeric(head, state)
7507     end
7508
7509     if outer == 'l' then
7510         if d == 'an' or d == 'r' then      -- im -> implicit
7511             if d == 'r' then state.has_r = true end
7512             state.sim = state.sim or item
7513             state.eim = item
7514         elseif d == 'l' and state.sim and state.has_r then
7515             head, state = insert_implicit(head, state, outer)
7516         elseif d == 'l' then
7517             state.sim, state.eim, state.has_r = nil, nil, false
7518         end
7519     else
7520         if d == 'an' or d == 'l' then
7521             if nodes[q][3] then -- nil except after an explicit dir
7522                 state.sim = item -- so we move sim 'inside' the group
7523             else
7524                 state.sim = state.sim or item
7525             end
7526         end
7527     end
7528 end

```



```

7526         state.eim = item
7527     elseif d == 'r' and state.sim then
7528         head, state = insert_implicit(head, state, outer)
7529     elseif d == 'r' then
7530         state.sim, state.eim = nil, nil
7531     end
7532 end
7533
7534 if isdir then
7535     last = d          -- Don't search back - best save now
7536 elseif d == 'on' and state.san then
7537     state.san = state.san or item
7538     state.ean = item
7539 end
7540
7541 end
7542
7543 head = node.prev(head) or head
7544
7545 ----- FIX HYPERLINKS -----
7546
7547 if has_hyperlink then
7548     local flag, linking = 0, 0
7549     for item in node.traverse(head) do
7550         if item.id == DIR then
7551             if item.dir == '+TRT' or item.dir == '+TLT' then
7552                 flag = flag + 1
7553             elseif item.dir == '-TRT' or item.dir == '-TLT' then
7554                 flag = flag - 1
7555             end
7556             elseif item.id == 8 and item.subtype == 19 then
7557                 linking = flag
7558             elseif item.id == 8 and item.subtype == 20 then
7559                 if linking > 0 then
7560                     if item.prev.id == DIR and
7561                         (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
7562                         d = node.new(DIR)
7563                         d.dir = item.prev.dir
7564                         node.remove(head, item.prev)
7565                         node.insert_after(head, item, d)
7566                     end
7567                 end
7568                 linking = 0
7569             end
7570         end
7571     end
7572
7573     return head
7574 end
7575 </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.

```
7576 < *nil>
7577 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language]
7578 \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.

```
7579 \ifx\l@nil\undefined
7580   \newlanguage\l@nil
7581   \@namedef{bbl@hyphendata@the\l@nil}{}{}{}% Remove warning
7582   \let\bbl@elt\relax
7583   \edef\bbl@languages{% Add it to the list of languages
7584     \bbl@languages\bbl@elt{nil}{the\l@nil}{}{}}
7585 \fi
```

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

```
7586 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
\datenil
7587 \let\captionnil\empty
7588 \let\datenil\empty
```

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

```
7589 \def\bbl@inidata@nil{%
7590   \bbl@elt{identification}{tag.ini}{und}%
7591   \bbl@elt{identification}{load.level}{0}%
7592   \bbl@elt{identification}{charset}{utf8}%
7593   \bbl@elt{identification}{version}{1.0}%
7594   \bbl@elt{identification}{date}{2022-05-16}%
7595   \bbl@elt{identification}{name.local}{nil}%
7596   \bbl@elt{identification}{name.english}{nil}%
7597   \bbl@elt{identification}{name.babel}{nil}%
7598   \bbl@elt{identification}{tag.bcp47}{und}%
7599   \bbl@elt{identification}{language.tag.bcp47}{und}%
7600   \bbl@elt{identification}{tag.opentype}{dflt}%
7601   \bbl@elt{identification}{script.name}{Latin}%
7602   \bbl@elt{identification}{script.tag.bcp47}{Latn}%
7603   \bbl@elt{identification}{script.tag.opentype}{DFLT}%
7604   \bbl@elt{identification}{level}{1}%
7605   \bbl@elt{identification}{encodings}{}%
7606   \bbl@elt{identification}{derivate}{no}}
7607 \@namedef{bbl@tbc@nil}{und}
7608 \@namedef{bbl@lbc@nil}{und}
7609 \@namedef{bbl@lotf@nil}{dflt}
7610 \@namedef{bbl@elname@nil}{nil}
7611 \@namedef{bbl@lname@nil}{nil}
7612 \@namedef{bbl@esname@nil}{Latin}
7613 \@namedef{bbl@sname@nil}{Latin}
7614 \@namedef{bbl@sbc@nil}{Latn}
7615 \@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.

```
7616 \ldf@finish{nil}
7617 </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.

```
7618 <<Compute Julian day>> ≡
7619 \def\bbl@fpmo#1#2{(#1-#2*floor(#1/#2))}
7620 \def\bbl@cs@gregleap#1{%
7621   (\bbl@fpmo{#1}{4} == 0) &&
7622   (!((\bbl@fpmo{#1}{100} == 0) && (\bbl@fpmo{#1}{400} != 0)))}
7623 \def\bbl@cs@jd#1#2#3{% year, month, day
7624   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
7625     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
7626     floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) +
7627     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3 }
7628 </Compute Julian day>>
```

### 15.1 Islamic

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

```
7629 <ca-islamic>
7630 \ExplSyntaxOn
7631 <<Compute Julian day>>
7632 % == islamic (default)
7633 % Not yet implemented
7634 \def\bbl@ca@islamic#1-#2-#3\@@#4#5#6{}
```

The Civil calendar:

```
7635 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
7636   ((#3 + ceil(29.5 * (#2 - 1)) +
7637     (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
7638     1948439.5) - 1) }
7639 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+2}}
7640 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
7641 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
7642 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
7643 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
7644 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@@#5#6#7{%
7645   \edef\bbl@tempa{%
7646     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1 }%
7647     \edef#5{%
7648       \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
7649     \edef#6{\fp_eval:n{
7650       min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
7651     \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).

```
7652 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
7653 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
7654 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
7655 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
7656 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
7657 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
7658 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
7659 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
7660 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
7661 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
7662 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
7663 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
```

```

7664 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
7665 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
7666 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
7667 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
7668 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
7669 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
7670 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
7671 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
7672 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
7673 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
7674 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
7675 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
7676 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
7677 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
7678 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
7679 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
7680 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
7681 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
7682 65401,65431,65460,65490,65520}
7683 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
7684 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
7685 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
7686 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
7687   \ifnum#2>2014 \ifnum#2<2038
7688     \bbl@afterfi\expandafter\@gobble
7689   \fi\fi
7690   {\bbl@error{Year~out-of-range}{The~allowed~range-is~2014-2038}}}%
7691 \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
7692   \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}}%
7693 \count@\@ne
7694 \bbl@foreach\bbl@cs@umalqura@data{%
7695   \advance\count@\@ne
7696   \ifnum##1>\bbl@tempd\else
7697     \edef\bbl@tempe{\the\count@}%
7698     \edef\bbl@tempb{##1}%
7699   \fi}%
7700 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month-lunar
7701 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1) / 12) }}% annus
7702 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
7703 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
7704 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
7705 \ExplSyntaxOff
7706 \bbl@add\bbl@precalendar{%
7707   \bbl@replace\bbl@ld@calendar{-civil}}}%
7708   \bbl@replace\bbl@ld@calendar{-umalqura}}}%
7709   \bbl@replace\bbl@ld@calendar{+}}}%
7710   \bbl@replace\bbl@ld@calendar{-}}}%
7711 \</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

```

7712 \<ca-hebrew>
7713 \newcount\bbl@cntcommon
7714 \def\bbl@remainder#1#2#3{%
7715   #3=#1\relax
7716   \divide #3 by #2\relax
7717   \multiply #3 by -#2\relax
7718   \advance #3 by #1\relax}%
7719 \newif\ifbbl@divisible

```

```

7720 \def\bbl@checkifdivisible#1#2{%
7721   {\countdef\tmp=0
7722     \bbl@remainder{#1}{#2}{\tmp}%
7723     \ifnum \tmp=0
7724       \global\bbl@divisibletrue
7725     \else
7726       \global\bbl@divisiblefalse
7727     \fi}}
7728 \newif\ifbbl@gregleap
7729 \def\bbl@ifgregleap#1{%
7730   \bbl@checkifdivisible{#1}{4}%
7731   \ifbbl@divisible
7732     \bbl@checkifdivisible{#1}{100}%
7733     \ifbbl@divisible
7734       \bbl@checkifdivisible{#1}{400}%
7735       \ifbbl@divisible
7736         \bbl@gregleaptrue
7737       \else
7738         \bbl@gregleapfalse
7739       \fi
7740     \else
7741       \bbl@gregleaptrue
7742     \fi
7743   \else
7744     \bbl@gregleapfalse
7745   \fi
7746   \ifbbl@gregleap}
7747 \def\bbl@gregdayspriormonths#1#2#3{%
7748   {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
7749     181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
7750   \bbl@ifgregleap{#2}%
7751   \ifnum #1 > 2
7752     \advance #3 by 1
7753   \fi
7754   \fi
7755   \global\bbl@cntcommon=#3}%
7756   #3=\bbl@cntcommon}
7757 \def\bbl@gregdaysprioryears#1#2{%
7758   {\countdef\tmpc=4
7759     \countdef\tmpb=2
7760     \tmpb=#1\relax
7761     \advance \tmpb by -1
7762     \tmpc=\tmpb
7763     \multiply \tmpc by 365
7764     #2=\tmpc
7765     \tmpc=\tmpb
7766     \divide \tmpc by 4
7767     \advance #2 by \tmpc
7768     \tmpc=\tmpb
7769     \divide \tmpc by 100
7770     \advance #2 by -\tmpc
7771     \tmpc=\tmpb
7772     \divide \tmpc by 400
7773     \advance #2 by \tmpc
7774     \global\bbl@cntcommon=#2\relax}%
7775   #2=\bbl@cntcommon}
7776 \def\bbl@absfromgreg#1#2#3#4{%
7777   {\countdef\tmpd=0
7778     #4=#1\relax
7779     \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
7780     \advance #4 by \tmpd
7781     \bbl@gregdaysprioryears{#3}{\tmpd}%
7782     \advance #4 by \tmpd

```

```

7783 \global\bbl@cntcommon=#4\relax}%
7784 #4=\bbl@cntcommon}
7785 \newif\ifbbl@hebrleap
7786 \def\bbl@checkleaphebrewyear#1{%
7787 {\countdef\tmpa=0
7788 \countdef\tmpb=1
7789 \tmpa=#1\relax
7790 \multiply \tmpa by 7
7791 \advance \tmpa by 1
7792 \bbl@remainder{\tmpa}{19}{\tmpb}%
7793 \ifnum \tmpb < 7
7794 \global\bbl@hebrleaptrue
7795 \else
7796 \global\bbl@hebrleapfalse
7797 \fi}}
7798 \def\bbl@hebreleapsedmonths#1#2{%
7799 {\countdef\tmpa=0
7800 \countdef\tmpb=1
7801 \countdef\tmpc=2
7802 \tmpa=#1\relax
7803 \advance \tmpa by -1
7804 #2=\tmpa
7805 \divide #2 by 19
7806 \multiply #2 by 235
7807 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
7808 \tmpc=\tmpb
7809 \multiply \tmpb by 12
7810 \advance #2 by \tmpb
7811 \multiply \tmpc by 7
7812 \advance \tmpc by 1
7813 \divide \tmpc by 19
7814 \advance #2 by \tmpc
7815 \global\bbl@cntcommon=#2}%
7816 #2=\bbl@cntcommon}
7817 \def\bbl@hebreleapseddays#1#2{%
7818 {\countdef\tmpa=0
7819 \countdef\tmpb=1
7820 \countdef\tmpc=2
7821 \bbl@hebreleapsedmonths{#1}{#2}%
7822 \tmpa=#2\relax
7823 \multiply \tmpa by 13753
7824 \advance \tmpa by 5604
7825 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
7826 \divide \tmpa by 25920
7827 \multiply #2 by 29
7828 \advance #2 by 1
7829 \advance #2 by \tmpa
7830 \bbl@remainder{#2}{7}{\tmpa}%
7831 \ifnum \tmpc < 19440
7832 \ifnum \tmpc < 9924
7833 \else
7834 \ifnum \tmpa=2
7835 \bbl@checkleaphebrewyear{#1}% of a common year
7836 \ifbbl@hebrleap
7837 \else
7838 \advance #2 by 1
7839 \fi
7840 \fi
7841 \fi
7842 \ifnum \tmpc < 16789
7843 \else
7844 \ifnum \tmpa=1
7845 \advance #1 by -1

```

```

7846             \bbl@checkleaphebyear{#1}% at the end of leap year
7847             \ifbbl@hebrleap
7848                 \advance #2 by 1
7849             \fi
7850         \fi
7851     \fi
7852 \else
7853     \advance #2 by 1
7854 \fi
7855 \bbl@remainder{#2}{7}{\tmpa}%
7856 \ifnum \tmpa=0
7857     \advance #2 by 1
7858 \else
7859     \ifnum \tmpa=3
7860         \advance #2 by 1
7861     \else
7862         \ifnum \tmpa=5
7863             \advance #2 by 1
7864         \fi
7865     \fi
7866 \fi
7867 \global\bbl@cntcommon=#2\relax}%
7868 #2=\bbl@cntcommon}
7869 \def\bbl@daysinhebyear#1#2{%
7870 {\countdef\tmpe=12
7871  \bbl@hebreleapseddays{#1}{\tmpe}%
7872  \advance #1 by 1
7873  \bbl@hebreleapseddays{#1}{#2}%
7874  \advance #2 by -\tmpe
7875  \global\bbl@cntcommon=#2}%
7876 #2=\bbl@cntcommon}
7877 \def\bbl@hebrdayspriormonths#1#2#3{%
7878 {\countdef\tmpf= 14
7879  #3=\ifcase #1\relax
7880      0 \or
7881      0 \or
7882      30 \or
7883      59 \or
7884      89 \or
7885      118 \or
7886      148 \or
7887      148 \or
7888      177 \or
7889      207 \or
7890      236 \or
7891      266 \or
7892      295 \or
7893      325 \or
7894      400
7895  \fi
7896  \bbl@checkleaphebyear{#2}%
7897  \ifbbl@hebrleap
7898      \ifnum #1 > 6
7899          \advance #3 by 30
7900      \fi
7901  \fi
7902  \bbl@daysinhebyear{#2}{\tmpf}%
7903  \ifnum #1 > 3
7904      \ifnum \tmpf=353
7905          \advance #3 by -1
7906      \fi
7907      \ifnum \tmpf=383
7908          \advance #3 by -1

```

```

7909     \fi
7910 \fi
7911 \ifnum #1 > 2
7912     \ifnum \tmpf=355
7913         \advance #3 by 1
7914     \fi
7915     \ifnum \tmpf=385
7916         \advance #3 by 1
7917     \fi
7918 \fi
7919 \global\bbl@cntcommon=#3\relax}%
7920 #3=\bbl@cntcommon}
7921 \def\bbl@absfromhebr#1#2#3#4{%
7922     {#4=#1\relax
7923     \bbl@hebrdayspriormonths{#2}{#3}{#1}%
7924     \advance #4 by #1\relax
7925     \bbl@hebreleapseddays{#3}{#1}%
7926     \advance #4 by #1\relax
7927     \advance #4 by -1373429
7928     \global\bbl@cntcommon=#4\relax}%
7929 #4=\bbl@cntcommon}
7930 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
7931     {\countdef\tmpx= 17
7932     \countdef\tmpy= 18
7933     \countdef\tmpz= 19
7934     #6=#3\relax
7935     \global\advance #6 by 3761
7936     \bbl@absfromgreg{#1}{#2}{#3}{#4}%
7937     \tmpz=1 \tmpy=1
7938     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7939     \ifnum \tmpx > #4\relax
7940         \global\advance #6 by -1
7941         \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7942     \fi
7943     \advance #4 by -\tmpx
7944     \advance #4 by 1
7945     #5=#4\relax
7946     \divide #5 by 30
7947     \loop
7948         \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
7949         \ifnum \tmpx < #4\relax
7950             \advance #5 by 1
7951             \tmpy=\tmpx
7952         \repeat
7953     \global\advance #5 by -1
7954     \global\advance #4 by -\tmpy}}
7955 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
7956 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
7957 \def\bbl@ca@hebrew#1-#2-#3@@#4#5#6{%
7958     \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
7959     \bbl@hebrfromgreg
7960     {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
7961     {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
7962     \edef#4{\the\bbl@hebryear}%
7963     \edef#5{\the\bbl@hebrmonth}%
7964     \edef#6{\the\bbl@hebrday}}
7965 \</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 LPPL is problematic. The code here follows loosely that by John Walker, which is free and accurate, but sadly very complex, so the relevant data for the years 2013-2050 have been pre-calculated and stored. Actually, all we need is the first day (either March 20 or March 21).

```

7966 <*ca-persian>
7967 \ExplSyntaxOn
7968 <<Compute Julian day>>
7969 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
7970 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
7971 \def\bbl@ca@persian#1-#2-#3\@#4#5#6{%
7972 \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
7973 \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
7974 \bbl@afterfi\expandafter\@gobble
7975 \fi\fi
7976 {\bbl@error{Year~out~of~range}{The~allowed~range~is~2013-2050}}%
7977 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
7978 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
7979 \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
7980 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
7981 \ifnum\bbl@tempc<\bbl@tempb
7982 \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
7983 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
7984 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
7985 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
7986 \fi
7987 \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
7988 \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
7989 \edef#5{\fp_eval:n{% set Jalali month
7990 (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
7991 \edef#6{\fp_eval:n{% set Jalali day
7992 (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6))}}
7993 \ExplSyntaxOff
7994 </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.

```

7995 <*ca-coptic>
7996 \ExplSyntaxOn
7997 <<Compute Julian day>>
7998 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
7999 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8000 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8001 \edef#4{\fp_eval:n{%
8002 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8003 \edef\bbl@tempc{\fp_eval:n{%
8004 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8005 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8006 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}
8007 \ExplSyntaxOff
8008 </ca-coptic>
8009 <*ca-ethiopic>
8010 \ExplSyntaxOn
8011 <<Compute Julian day>>
8012 \def\bbl@ca@ethiopic#1-#2-#3\@#4#5#6{%
8013 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8014 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8015 \edef#4{\fp_eval:n{%
8016 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8017 \edef\bbl@tempc{\fp_eval:n{%
8018 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%

```

```

8019 \edef#5{\fp_eval:n{\floor(\bbl@tempc / 30) + 1}}%
8020 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}%
8021 \ExplSyntaxOff
8022 \</ca-ethiopic>

```

## 19 Buddhist

That's very simple.

```

8023 \<ca-buddhist>
8024 \def\bbl@ca@buddhist#1-#2-#3\@#4#5#6{%
8025 \edef#4{\number\numexpr#1+543\relax}%
8026 \edef#5{#2}%
8027 \edef#6{#3}}
8028 \</ca-buddhist>

```

## 20 Support for Plain T<sub>E</sub>X (plain.def)

### 20.1 Not renaming hyphen.tex

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based T<sub>E</sub>X-format. When asked he responded:

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

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

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to achieve the desired effect, based on the `babel` package. If you load each of them with `iniTEX`, you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`.

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

```

8029 \<bplain | blplain>
8030 \catcode`\{=1 % left brace is begin-group character
8031 \catcode`\}=2 % right brace is end-group character
8032 \catcode`\#=6 % hash mark is macro parameter character

```

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

```

8033 \openin 0 hyphen.cfg
8034 \ifeof0
8035 \else
8036 \let\a\input

```

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

```

8037 \def\input #1 {%
8038 \let\input\a
8039 \a hyphen.cfg
8040 \let\a\undefined
8041 }
8042 \fi
8043 \<bplain | blplain>

```

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

```

8044 \<bplain>\a plain.tex
8045 \<blplain>\a lplain.tex

```

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

```
8046 \bplain\def\fmtname{babel-plain}
8047 \blplain\def\fmtname{babel-lplain}
```

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

## 20.2 Emulating some $\text{\LaTeX}$ features

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

```
8048 \langle *Emulate LaTeX \rangle \equiv
8049 \def\@empty{}
8050 \def\loadlocalcfg#1{%
8051   \openin0#1.cfg
8052   \ifeof0
8053     \closein0
8054   \else
8055     \closein0
8056     {\immediate\write16{*****}%
8057      \immediate\write16{* Local config file #1.cfg used}%
8058      \immediate\write16{*}%
8059     }
8060     \input #1.cfg\relax
8061   \fi
8062   \@endofldf}
```

## 20.3 General tools

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

```
8063 \long\def\@firstofone#1{#1}
8064 \long\def\@firstoftwo#1#2{#1}
8065 \long\def\@secondoftwo#1#2{#2}
8066 \def\@nnil{\nil}
8067 \def\@gobbletwo#1#2{}
8068 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
8069 \def\@star@or@long#1{%
8070   \@ifstar
8071   {\let\@ngrel@x\relax#1}%
8072   {\let\@ngrel@x\long#1}}
8073 \let\@ngrel@x\relax
8074 \def\@car#1#2\@nil{#1}
8075 \def\@cdr#1#2\@nil{#2}
8076 \let\@typeset@protect\relax
8077 \let\protected@edef\edef
8078 \long\def\@gobble#1{}
8079 \edef\@backslashchar{\expandafter\@gobble\string\}
8080 \def\strip@prefix#1>{}
8081 \def\g@addto@macro#1#2{{%
8082   \toks@\expandafter{#1#2}%
8083   \xdef#1{\the\toks@}}}
8084 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8085 \def\@nameuse#1{\csname #1\endcsname}
8086 \def\@ifundefined#1{%
8087   \expandafter\ifx\csname#1\endcsname\relax
8088     \expandafter\@firstoftwo
8089   \else
8090     \expandafter\@secondoftwo
```

```

8091 \fi}
8092 \def\@expandtwoargs#1#2#3{%
8093 \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8094 \def\zap@space#1 #2{%
8095 #1%
8096 \ifx#2\@empty\else\expandafter\zap@space\fi
8097 #2}
8098 \let\bbl@trace\@gobble
8099 \def\bbl@error#1#2{%
8100 \begingroup
8101 \newlinechar=`^^J
8102 \def\{^^J(babel) }%
8103 \errhelp{#2}\errmessage{\#1}%
8104 \endgroup}
8105 \def\bbl@warning#1{%
8106 \begingroup
8107 \newlinechar=`^^J
8108 \def\{^^J(babel) }%
8109 \message{\#1}%
8110 \endgroup}
8111 \let\bbl@infowarn\bbl@warning
8112 \def\bbl@info#1{%
8113 \begingroup
8114 \newlinechar=`^^J
8115 \def\{^^J}%
8116 \wlog{#1}%
8117 \endgroup}

```

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

```

8118 \ifx\@preamblecmds\undefined
8119 \def\@preamblecmds{}
8120 \fi
8121 \def\@onlypreamble#1{%
8122 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8123 \@preamblecmds\do#1}}
8124 \@onlypreamble\@onlypreamble

```

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

```

8125 \def\begin{document}{%
8126 \@begin{document}hook
8127 \global\let\@begin{document}hook\undefined
8128 \def\do##1{\global\let##1\undefined}%
8129 \@preamblecmds
8130 \global\let\do\noexpand}

8131 \ifx\@begin{document}hook\undefined
8132 \def\@begin{document}hook{}
8133 \fi
8134 \@onlypreamble\@begin{document}hook
8135 \def\AtBeginDocument{\g@addto@macro\@begin{document}hook}

```

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

```

8136 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
8137 \@onlypreamble\AtEndOfPackage
8138 \def\@endoflfd{}
8139 \@onlypreamble\@endoflfd
8140 \let\bbl@afterlang\@empty
8141 \chardef\bbl@opt@hyphenmap\z@

```

$\text{\LaTeX}$  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.

```

8142 \catcode`\&=\z@
8143 \ifx&\if@files\@undefined
8144   \expandafter\let\csname if@files\expandafter\endcsname
8145     \csname iffalse\endcsname
8146 \fi
8147 \catcode`\&=4

```

Mimick L<sup>A</sup>T<sub>E</sub>X's commands to define control sequences.

```

8148 \def\newcommand{\@star@or@long\new@command}
8149 \def\new@command#1{%
8150   \@testopt{\@newcommand#1}0}
8151 \def\@newcommand#1[#2]{%
8152   \@ifnextchar [{\@xargdef#1[#2]}%
8153     {\@argdef#1[#2]}}
8154 \long\def\@argdef#1[#2]#3{%
8155   \@yargdef#1\@ne{#2}{#3}}
8156 \long\def\@xargdef#1[#2][#3]#4{%
8157   \expandafter\def\expandafter#1\expandafter{%
8158     \expandafter\@protected@testopt\expandafter #1%
8159     \csname\string#1\expandafter\endcsname{#3}}%
8160   \expandafter\@yargdef \csname\string#1\endcsname
8161     \tw@{#2}{#4}}
8162 \long\def\@yargdef#1#2#3{%
8163   \@tempcnta#3\relax
8164   \advance \@tempcnta \@ne
8165   \let\@hash@\relax
8166   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
8167   \@tempcntb #2%
8168   \@whilenum\@tempcntb <\@tempcnta
8169   \do{%
8170     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
8171     \advance\@tempcntb \@ne}%
8172   \let\@hash@##%
8173   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
8174 \def\providecommand{\@star@or@long\provide@command}
8175 \def\provide@command#1{%
8176   \begingroup
8177     \escapechar\m@ne\def\@gtempa{\string#1}%
8178   \endgroup
8179   \expandafter\@ifundefined\@gtempa
8180     {\def\reserved@a{\new@command#1}}%
8181     {\let\reserved@a\relax
8182      \def\reserved@a{\new@command\reserved@a}}%
8183   \reserved@a}%
8184 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
8185 \def\declare@robustcommand#1{%
8186   \edef\reserved@a{\string#1}%
8187   \def\reserved@b{#1}%
8188   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@a}%
8189   \edef#1{%
8190     \ifx\reserved@a\reserved@b
8191       \noexpand\x@protect
8192       \noexpand#1%
8193     \fi
8194     \noexpand\protect
8195     \expandafter\noexpand\csname
8196       \expandafter\@gobble\string#1 \endcsname
8197   }%
8198   \expandafter\new@command\csname
8199     \expandafter\@gobble\string#1 \endcsname
8200 }
8201 \def\x@protect#1{%
8202   \ifx\protect\@typeset@protect\else

```

```

8203      \@x@protect#1%
8204      \fi
8205 }
8206 \catcode\&=\z@ % Trick to hide conditionals
8207 \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`.

```

8208 \def\bbl@tempa{\csname newif\endcsname&fin@}
8209 \catcode\&=4
8210 \ifx\in@\@undefined
8211 \def\in@#1#2{%
8212 \def\in@@##1##2##3\in@{%
8213 \ifx\in@@##2\in@false\else\in@true\fi}%
8214 \in@@##1\in@\in@@}
8215 \else
8216 \let\bbl@tempa\@empty
8217 \fi
8218 \bbl@tempa

```

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

```

8219 \def\@ifpackagewith#1#2#3#4{#3}

```

The  $\TeX$  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.

```

8220 \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  $\TeX 2_{\epsilon}$  versions; just enough to make things work in plain  $\TeX$  environments.

```

8221 \ifx\@tempcnta\@undefined
8222 \csname newcount\endcsname\@tempcnta\relax
8223 \fi
8224 \ifx\@tempcntb\@undefined
8225 \csname newcount\endcsname\@tempcntb\relax
8226 \fi

```

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

```

8227 \ifx\bye\@undefined
8228 \advance\count10 by -2\relax
8229 \fi
8230 \ifx\@ifnextchar\@undefined
8231 \def\@ifnextchar#1#2#3{%
8232 \let\reserved@d=#1%
8233 \def\reserved@a{#2}\def\reserved@b{#3}%
8234 \futurelet\@let@token\@ifnch}
8235 \def\@ifnch{%
8236 \ifx\@let@token\@sptoken
8237 \let\reserved@c\@xifnch
8238 \else
8239 \ifx\@let@token\reserved@d
8240 \let\reserved@c\reserved@a
8241 \else
8242 \let\reserved@c\reserved@b
8243 \fi
8244 \fi
8245 \reserved@c}
8246 \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token

```

```

8247 \def\{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
8248 \fi
8249 \def\@testopt#1#2{%
8250 \ifnextchar[{\#1}{\#1[#2]}}
8251 \def\@protected@testopt#1{%
8252 \ifx\protect\@typeset@protect
8253 \expandafter\@testopt
8254 \else
8255 \@x@protect#1%
8256 \fi}
8257 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8258 #2\relax}\fi}
8259 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8260 \else\expandafter\@gobble\fi{#1}}

```

## 20.4 Encoding related macros

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

```

8261 \def\DeclareTextCommand{%
8262 \@dec@text@cmd\providecommand
8263 }
8264 \def\ProvideTextCommand{%
8265 \@dec@text@cmd\providecommand
8266 }
8267 \def\DeclareTextSymbol#1#2#3{%
8268 \@dec@text@cmd\chardef#1{#2}#3\relax
8269 }
8270 \def\@dec@text@cmd#1#2#3{%
8271 \expandafter\def\expandafter#2%
8272 \expandafter{%
8273 \csname#3-cmd\expandafter\endcsname
8274 \expandafter#2%
8275 \csname#3\string#2\endcsname
8276 }%
8277 % \let\@ifdefinable\@rc@ifdefinable
8278 \expandafter#1\csname#3\string#2\endcsname
8279 }
8280 \def\@current@cmd#1{%
8281 \ifx\protect\@typeset@protect\else
8282 \noexpand#1\expandafter\@gobble
8283 \fi
8284 }
8285 \def\@changed@cmd#1#2{%
8286 \ifx\protect\@typeset@protect
8287 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8288 \expandafter\ifx\csname ?\string#1\endcsname\relax
8289 \expandafter\def\csname ?\string#1\endcsname{%
8290 \@changed@x@err{#1}%
8291 }%
8292 \fi
8293 \global\expandafter\let
8294 \csname\cf@encoding\string#1\expandafter\endcsname
8295 \csname ?\string#1\endcsname
8296 \fi
8297 \csname\cf@encoding\string#1%
8298 \expandafter\endcsname
8299 \else
8300 \noexpand#1%
8301 \fi
8302 }
8303 \def\@changed@x@err#1{%
8304 \errhelp{Your command will be ignored, type <return> to proceed}%
8305 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}

```

```

8306 \def\DeclareTextCommandDefault#1{%
8307   \DeclareTextCommand#1?%
8308 }
8309 \def\ProvideTextCommandDefault#1{%
8310   \ProvideTextCommand#1?%
8311 }
8312 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8313 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8314 \def\DeclareTextAccent#1#2#3{%
8315   \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
8316 }
8317 \def\DeclareTextCompositeCommand#1#2#3#4{%
8318   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8319   \edef\reserved@b{\string##1}%
8320   \edef\reserved@c{%
8321     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8322   \ifx\reserved@b\reserved@c
8323     \expandafter\expandafter\expandafter\ifx
8324       \expandafter\@car\reserved@a\relax\relax\@nil
8325       \@text@composite
8326   \else
8327     \edef\reserved@b##1{%
8328       \def\expandafter\noexpand
8329         \csname#2\string#1\endcsname####1{%
8330         \noexpand\@text@composite
8331         \expandafter\noexpand\csname#2\string#1\endcsname
8332         ####1\noexpand\@empty\noexpand\@text@composite
8333         {##1}%
8334       }%
8335     }%
8336     \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8337   \fi
8338   \expandafter\def\csname\expandafter\string\csname
8339     #2\endcsname\string#1-\string#3\endcsname{#4}
8340 \else
8341   \errhelp{Your command will be ignored, type <return> to proceed}%
8342   \errmessage{\string\DeclareTextCompositeCommand\space used on
8343     inappropriate command \protect#1}
8344 \fi
8345 }
8346 \def\@text@composite#1#2#3\@text@composite{%
8347   \expandafter\@text@composite@x
8348     \csname\string#1-\string#2\endcsname
8349 }
8350 \def\@text@composite@x#1#2{%
8351   \ifx#1\relax
8352     #2%
8353   \else
8354     #1%
8355   \fi
8356 }
8357 %
8358 \def\@strip@args#1:#2-#3\@strip@args{#2}
8359 \def\DeclareTextComposite#1#2#3#4{%
8360   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8361   \bgroup
8362     \lccode`\@=#4%
8363     \lowercase{%
8364       \egroup
8365       \reserved@a @%
8366     }%
8367 }
8368 %

```



```

8369 \def\UseTextSymbol#1#2{#2}
8370 \def\UseTextAccent#1#2#3{}
8371 \def\@use@text@encoding#1{}
8372 \def\DeclareTextSymbolDefault#1#2{%
8373   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8374 }
8375 \def\DeclareTextAccentDefault#1#2{%
8376   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8377 }
8378 \def\cf@encoding{OT1}

```

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

```

8379 \DeclareTextAccent{"}{OT1}{127}
8380 \DeclareTextAccent{'}{OT1}{19}
8381 \DeclareTextAccent{^}{OT1}{94}
8382 \DeclareTextAccent{\`}{OT1}{18}
8383 \DeclareTextAccent{\~}{OT1}{126}

```

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

```

8384 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8385 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
8386 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
8387 \DeclareTextSymbol{\textquoteright}{OT1}{`'}
8388 \DeclareTextSymbol{\i}{OT1}{16}
8389 \DeclareTextSymbol{\ss}{OT1}{25}

```

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

```

8390 \ifx\scriptsize\@undefined
8391   \let\scriptsize\sevenrm
8392 \fi

```

And a few more “dummy” definitions.

```

8393 \def\language#1{english}%
8394 \let\bbl@opt@shorthands\@nnil
8395 \def\bbl@ifshorthand#1#2#3#2{%
8396   \let\bbl@language@opts\@empty
8397   \ifx\babeloptionstrings\@undefined
8398     \let\bbl@opt@strings\@nnil
8399   \else
8400     \let\bbl@opt@strings\babeloptionstrings
8401   \fi
8402 \def\BabelStringsDefault{generic}
8403 \def\bbl@tempa{normal}
8404 \ifx\babeloptionmath\bbl@tempa
8405   \def\bbl@mathnormal{\noexpand\textormath}
8406 \fi
8407 \def\AfterBabelLanguage#1#2{}
8408 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
8409 \let\bbl@afterlang\relax
8410 \def\bbl@opt@safe{BR}
8411 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
8412 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
8413 \expandafter\newif\csname ifbbl@single\endcsname
8414 \chardef\bbl@bidimode\z@
8415 <</Emulate LaTeX>>

```

A proxy file:

```

8416 <*plain>
8417 \input babel.def
8418 </plain>

```

## 21 Acknowledgements

I would like to thank all who volunteered as  $\beta$ -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs.

During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

## References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national  $\text{\LaTeX}$  styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The  $\text{\TeX}$ book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport,  *$\text{\LaTeX}$ , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in:  $\text{\TeX}$ hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018
- [10] Hubert Partl, *German  $\text{\TeX}$* , *TUGboat* 9 (1988) #1, p. 70–72.
- [11] Joachim Schrod, *International  $\text{\LaTeX}$  is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using  $\text{\LaTeX}$* , Springer, 2002, p. 301–373.
- [13] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).