

# Babel

Version 3.60.2416  
2021/06/27

Johannes L. Braams  
Original author

Javier Bezos  
Current maintainer

Localization and  
internationalization

Unicode

T<sub>E</sub>X

pdfT<sub>E</sub>X

LuaT<sub>E</sub>X

XeT<sub>E</sub>X

# 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 . . . . .	8
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 . . . . .	26
1.15	Modifying a language . . . . .	28
1.16	Creating a language . . . . .	29
1.17	Digits and counters . . . . .	33
1.18	Dates . . . . .	34
1.19	Accessing language info . . . . .	35
1.20	Hyphenation and line breaking . . . . .	36
1.21	Transforms . . . . .	38
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 . . . . .	49
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 . . . . .	57
3.6	Support for extending macros . . . . .	57
3.7	Macros common to a number of languages . . . . .	57
3.8	Encoding-dependent strings . . . . .	57
<b>4</b>	<b>Changes</b>	<b>61</b>
4.1	Changes in babel version 3.9 . . . . .	61

<b>II</b>	<b>Source code</b>	<b>62</b>
<b>5</b>	<b>Identification and loading of required files</b>	<b>62</b>
<b>6</b>	<b>locale directory</b>	<b>62</b>
<b>7</b>	<b>Tools</b>	<b>63</b>
7.1	Multiple languages . . . . .	67
7.2	The Package File ( <code>\LaTeX</code> , <code>babel.sty</code> ) . . . . .	67
7.3	base . . . . .	69
7.4	Conditional loading of shorthands . . . . .	72
7.5	Cross referencing macros . . . . .	73
7.6	Marks . . . . .	76
7.7	Preventing clashes with other packages . . . . .	77
7.7.1	ifthen . . . . .	77
7.7.2	varioref . . . . .	77
7.7.3	hhline . . . . .	78
7.7.4	hyperref . . . . .	78
7.7.5	fancyhdr . . . . .	78
7.8	Encoding and fonts . . . . .	79
7.9	Basic bidi support . . . . .	81
7.10	Local Language Configuration . . . . .	86
7.11	Language options . . . . .	86
<b>8</b>	<b>The kernel of Babel (<code>babel.def</code>, <code>common</code>)</b>	<b>90</b>
8.1	Tools . . . . .	90
<b>9</b>	<b>Multiple languages</b>	<b>91</b>
9.1	Selecting the language . . . . .	93
9.2	Errors . . . . .	102
9.3	Hooks . . . . .	105
9.4	Setting up language files . . . . .	106
9.5	Shorthands . . . . .	108
9.6	Language attributes . . . . .	118
9.7	Support for saving macro definitions . . . . .	120
9.8	Short tags . . . . .	121
9.9	Hyphens . . . . .	121
9.10	Multiencoding strings . . . . .	123
9.11	Macros common to a number of languages . . . . .	130
9.12	Making glyphs available . . . . .	130
9.12.1	Quotation marks . . . . .	130
9.12.2	Letters . . . . .	131
9.12.3	Shorthands for quotation marks . . . . .	132
9.12.4	Umlauts and tremas . . . . .	133
9.13	Layout . . . . .	134
9.14	Load engine specific macros . . . . .	135
9.15	Creating and modifying languages . . . . .	135
<b>10</b>	<b>Adjusting the Babel behavior</b>	<b>156</b>
<b>11</b>	<b>Loading hyphenation patterns</b>	<b>158</b>
<b>12</b>	<b>Font handling with fontspec</b>	<b>163</b>

<b>13</b>	<b>Hooks for XeTeX and LuaTeX</b>	<b>167</b>
13.1	XeTeX . . . . .	167
13.2	Layout . . . . .	169
13.3	LuaTeX . . . . .	171
13.4	Southeast Asian scripts . . . . .	177
13.5	CJK line breaking . . . . .	178
13.6	Arabic justification . . . . .	180
13.7	Common stuff . . . . .	184
13.8	Automatic fonts and ids switching . . . . .	185
13.9	Layout . . . . .	198
13.10	Auto bidi with basic and basic-r . . . . .	202
<b>14</b>	<b>Data for CJK</b>	<b>212</b>
<b>15</b>	<b>The ‘nil’ language</b>	<b>213</b>
<b>16</b>	<b>Support for Plain T<sub>E</sub>X (plain.def)</b>	<b>213</b>
16.1	Not renaming hyphen.tex . . . . .	213
16.2	Emulating some L <sub>A</sub> T <sub>E</sub> X features . . . . .	214
16.3	General tools . . . . .	215
16.4	Encoding related macros . . . . .	218
<b>17</b>	<b>Acknowledgements</b>	<b>221</b>

## Troubleshooting

Paragraph ended before \UTFviii@three@octets was complete . . . . .	5
No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the format . . . . .	6
You are loading directly a language style . . . . .	8
Unknown language ‘LANG’ . . . . .	9
Argument of \language@active@arg” has an extra } . . . . .	12
Package fontspec Warning: ‘Language ‘LANG’ not available for font ‘FONT’ with script ‘SCRIPT’ ‘Default’ language used instead’ . . . . .	28
Package babel Info: The following fonts are not babel standard families . . . . .	28

# Part I

## User guide

**What is this document about?** This user guide focuses on internationalization and localization with  $\LaTeX$  and `pdftex`, `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:

PDFTEX

```
\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.

## 1.2 Multilingual documents

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

**EXAMPLE** In  $\text{\LaTeX}$ , the preamble of the document:

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

would tell  $\text{\LaTeX}$  that the document would be written in two languages, Dutch and English, and that English would be the first language in use, and the main one.

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

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

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

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

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

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

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

**WARNING** In the preamble the main language has *not* been selected, except hyphenation patterns and the name assigned to `\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 pdfTeX follows. The main language is french, which is activated when the document begins. It assumes UTF-8:

PDFTEX

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

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

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, `yi`). See section 1.22 for further details.

### 1.4 Modifiers

**New 3.9c** The basic behavior of some languages can be modified when loading babel by means of *modifiers*. They are set after the language name, and are prefixed with a dot (only when the language is set as package option – neither global options nor the main key accepts them). An example is (spaces are not significant and they can be added or removed):<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 L<sup>A</sup>T<sub>E</sub>X (ie, `\usepackage{⟨language⟩}`) is deprecated and you will get the error:<sup>2</sup>

<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.

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

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

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

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

## 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.

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

**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.

**\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 `other language*` does not.

## 1.9 More on selection

**`\babeltags`**  $\langle tag1 \rangle = \langle language1 \rangle, \langle tag2 \rangle = \langle language2 \rangle, \dots$

**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{\langle language1 \rangle}\{<text>\}`, and `\begin{\langle tag1 \rangle}` to be `\begin{other language*}\{\langle language1 \rangle\}`, and so on. Note `\langle tag1 \rangle` is also allowed, but remember to set it locally inside a group.

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

**EXAMPLE** With

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

you can write

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

and

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

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

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

**`\babelensure`**  $[\text{include}=\langle commands \rangle, \text{exclude}=\langle commands \rangle, \text{fontenc}=\langle encoding \rangle]\{\langle language \rangle\}$

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

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

## 1.10 Shorthands

A *shorthand* is a sequence of one or two characters that expands to arbitrary  $\TeX$  code. Shorthands can be used for different kinds of things; for example: (1) in some languages shorthands such as "a are defined to be able to hyphenate the word if the encoding is OT1; (2) in some languages shorthands such as ! are used to insert the right amount of white space; (3) several kinds of discretionary breaks can be inserted easily with "-", "=", etc. The package `inputenc` as well as `xetex` and `luatex` have alleviated entering non-ASCII characters, but minority languages and some kinds of text can still require characters not directly available on the keyboards (and sometimes not even as separated or precomposed Unicode characters). As to the point 2, now `pdfTeX` provides `\knbccode`, and `luatex` can manipulate the glyph list. Tools for point 3 can be still very useful in general. There are four levels of shorthands: *user*, *language*, *system*, and *language user* (by order of precedence). In most cases, you will use only shorthands provided by languages.

**NOTE** Keep in mind the following:

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

**TROUBLESHOOTING** A typical error when using shorthands is the following:

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

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

`\shorthandon`  $\{ \langle shorthands-list \rangle \}$

**\shorthandoff** `*{\<shorthands-list>}`

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

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

```
\shorthandoff*{~^}
```

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

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

**\usesshorthands** `*{\<char>}`

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

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

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

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

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

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

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

**EXAMPLE** Let’s assume you want a unified set of shorthand for 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:

---

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

```
\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

<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

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

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

In addition, the babel core declares ~ as a one-char shorthand which is let, like the standard ~, to a non breaking space.<sup>7</sup>

**\ifbabelshorthand** {<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.



<b>KeepShorthandsActive</b>	Tells babel not to deactivate shorthands after loading a language file, so that they are also available in the preamble.
<b>activeacute</b>	For some languages babel supports this options to set ' as a shorthand in case it is not done by default.
<b>activegrave</b>	Same for `.
<b>shorthands=</b>	<p><math>\langle char \rangle \langle char \rangle \dots</math>   off</p> <p>The only language shorthands activated are those given, like, eg:</p> <pre>\usepackage[esperanto,french,shorthands=;!]{babel}</pre> <p>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 <math>\LaTeX</math> 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.</p>
<b>safe=</b>	<p>none   ref   bib</p> <p>Some <math>\LaTeX</math> 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 <b>New 3.34</b>, in <math>\epsilon\TeX</math> based engines (ie, almost every engine except the oldest ones) shorthands can be used in these macros (formerly you could not).</p>
<b>math=</b>	<p>active   normal</p> <p>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 <math>\{a'\}</math> (a closing brace after a shorthand) are not a source of trouble anymore.</p>
<b>config=</b>	<p><math>\langle file \rangle</math></p> <p>Load <math>\langle file \rangle.cfg</math> instead of the default config file bblopts.cfg (the file is loaded even with noconfigs).</p>
<b>main=</b>	<p><math>\langle language \rangle</math></p> <p>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.</p>
<b>headfoot=</b>	<p><math>\langle language \rangle</math></p> <p>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.</p>

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

<b>noconfigs</b>	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.
<b>showlanguages</b>	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.
<b>nocase</b>	<b>New 3.9l</b> Language settings for uppercase and lowercase mapping (as set by \SetCase) are ignored. Use only if there are incompatibilities with other packages.
<b>silent</b>	<b>New 3.9l</b> No warnings and no <i>infos</i> are written to the log file. <sup>8</sup>
<b>strings=</b>	generic   unicode   encoded   <i>&lt;label&gt;</i>   <i>&lt;font encoding&gt;</i> Selects the encoding of strings in languages supporting this feature. Predefined labels are generic (for traditional T <sub>E</sub> X, L <sup>A</sup> T <sub>E</sub> X and ASCII strings), unicode (for engines like xetex and luatex) and encoded (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in \MakeUppercase and the like (this feature misuses some internal L <sup>A</sup> T <sub>E</sub> X tools, so use it only as a last resort).
<b>hyphenmap=</b>	off   first   select   other   other* <b>New 3.9g</b> Sets the behavior of case mapping for hyphenation, provided the language defines it. <sup>9</sup> It can take the following values: <b>off</b> deactivates this feature and no case mapping is applied; <b>first</b> 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> <b>select</b> sets it only at \selectlanguage; <b>other</b> also sets it at otherlanguage; <b>other*</b> also sets it at otherlanguage* as well as in heads and foots (if the option headfoot is used) and in auxiliary files (ie, at \select@language), and it's the default if several language options have been stated. The option first can be regarded as an optimized version of other* for monolingual documents. <sup>11</sup>
<b>bidi=</b>	default   basic   basic-r   bidi-l   bidi-r <b>New 3.14</b> Selects the bidi algorithm to be used in luatex and xetex. See sec. 1.24.
<b>layout=</b>	<b>New 3.16</b> Selects which layout elements are adapted in bidi documents. See sec. 1.24.

## 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

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

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

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

<sup>11</sup>Providing foreign is pointless, because the case mapping applied is that at the end of the paragraph, but if either xetex or luatex change this behavior it might be added. On the other hand, other is provided even if I [JBL] think it isn't really useful, but who knows.

last language passed as option (by its name in `language.dat`). There are two main uses: classes and packages, and as a last resort in case there are, for some reason, incompatible languages. It can be used if you just want to select the hyphenation patterns of a single language, too.

`\AfterBabelLanguage`  $\langle option-name \rangle \{ \langle code \rangle \}$

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

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

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

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

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

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

### 1.13 ini files

An alternative approach to define a language (or, more precisely, a *locale*) is by means of an `ini` file. Currently babel provides about 200 of these files containing the basic data required for a locale.

`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{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}

```

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

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

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

```

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

```

Or also:

```

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

```

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

```

\babelfont[spanish]{rm}{FreeSerif}
\babelfont[hindi]{rm}[Renderer=Harfbuzz]{FreeSerif}

```

**Arabic** Monolingual documents mostly work in `luatex`, but it must be fine tuned, particularly 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, but still problematic).

**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 can be modified in luatex; they are hard-coded in xetex). Lao seems to work, too, but there are no patterns for the latter in luatex. Khemer clusters are rendered wrongly with the default renderer. The comment about Indic scripts and lualatex also applies here. Some quick patterns can help, with something similar to:

```
\babelprovide[import, hyphenrules=+]{lao}
\babelpatterns[lao]{lᦺ lᦴ lᦳ lᦵ lᦶ lᦷ} % 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 seems 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>	bo	Tibetan <sup>u</sup>
agq	Aghem	brx	Bodo
ak	Akan	bs-Cyrl	Bosnian
am	Amharic <sup>ul</sup>	bs-Latn	Bosnian <sup>ul</sup>
ar	Arabic <sup>ul</sup>	bs	Bosnian <sup>ul</sup>
ar-DZ	Arabic <sup>ul</sup>	ca	Catalan <sup>ul</sup>
ar-MA	Arabic <sup>ul</sup>	ce	Chechen
ar-SY	Arabic <sup>ul</sup>	cgg	Chiga
as	Assamese	chr	Cherokee
asa	Asu	ckb	Central Kurdish
ast	Asturian <sup>ul</sup>	cop	Coptic
az-Cyrl	Azerbaijani	cs	Czech <sup>ul</sup>
az-Latn	Azerbaijani	cu	Church Slavic
az	Azerbaijani <sup>ul</sup>	cu-Cyrs	Church Slavic
bas	Basaa	cu-Glag	Church Slavic
be	Belarusian <sup>ul</sup>	cy	Welsh <sup>ul</sup>
bem	Bemba	da	Danish <sup>ul</sup>
bez	Bena	dav	Taita
bg	Bulgarian <sup>ul</sup>	de-AT	German <sup>ul</sup>
bm	Bambara	de-CH	German <sup>ul</sup>
bn	Bangla <sup>ul</sup>	de	German <sup>ul</sup>

dje	Zarma	ii	Sichuan Yi
dsb	Lower Sorbian <sup>ul</sup>	is	Icelandic <sup>ul</sup>
dua	Duala	it	Italian <sup>ul</sup>
dyo	Jola-Fonyi	ja	Japanese
dz	Dzongkha	jgo	Ngomba
ebu	Embu	jmc	Machame
ee	Ewe	ka	Georgian <sup>ul</sup>
el	Greek <sup>ul</sup>	kab	Kabyle
el-polyton	Polytonic Greek <sup>ul</sup>	kam	Kamba
en-AU	English <sup>ul</sup>	kde	Makonde
en-CA	English <sup>ul</sup>	kea	Kabuverdianu
en-GB	English <sup>ul</sup>	khq	Koyra Chiini
en-NZ	English <sup>ul</sup>	ki	Kikuyu
en-US	English <sup>ul</sup>	kk	Kazakh
en	English <sup>ul</sup>	kkj	Kako
eo	Esperanto <sup>ul</sup>	kl	Kalaallisut
es-MX	Spanish <sup>ul</sup>	kln	Kalenjin
es	Spanish <sup>ul</sup>	km	Khmer
et	Estonian <sup>ul</sup>	kn	Kannada <sup>ul</sup>
eu	Basque <sup>ul</sup>	ko	Korean
ewo	Ewondo	kok	Konkani
fa	Persian <sup>ul</sup>	ks	Kashmiri
ff	Fulah	ksb	Shambala
fi	Finnish <sup>ul</sup>	ksf	Bafia
fil	Filipino	ksh	Colognian
fo	Faroese	kw	Cornish
fr	French <sup>ul</sup>	ky	Kyrgyz
fr-BE	French <sup>ul</sup>	lag	Langi
fr-CA	French <sup>ul</sup>	lb	Luxembourgish
fr-CH	French <sup>ul</sup>	lg	Ganda
fr-LU	French <sup>ul</sup>	lkt	Lakota
fur	Friulian <sup>ul</sup>	ln	Lingala
fy	Western Frisian	lo	Lao <sup>ul</sup>
ga	Irish <sup>ul</sup>	lrc	Northern Luri
gd	Scottish Gaelic <sup>ul</sup>	lt	Lithuanian <sup>ul</sup>
gl	Galician <sup>ul</sup>	lu	Luba-Katanga
grc	Ancient Greek <sup>ul</sup>	luo	Luo
gsw	Swiss German	luy	Luyia
gu	Gujarati	lv	Latvian <sup>ul</sup>
guz	Gusii	mas	Masai
gv	Manx	mer	Meru
ha-GH	Hausa	mfe	Morisyen
ha-NE	Hausa <sup>1</sup>	mg	Malagasy
ha	Hausa	mgh	Makhuwa-Meetto
haw	Hawaiian	mgo	Meta'
he	Hebrew <sup>ul</sup>	mk	Macedonian <sup>ul</sup>
hi	Hindi <sup>u</sup>	ml	Malayalam <sup>ul</sup>
hr	Croatian <sup>ul</sup>	mn	Mongolian
hsb	Upper Sorbian <sup>ul</sup>	mr	Marathi <sup>ul</sup>
hu	Hungarian <sup>ul</sup>	ms-BN	Malay <sup>1</sup>
hy	Armenian <sup>u</sup>	ms-SG	Malay <sup>1</sup>
ia	Interlingua <sup>ul</sup>	ms	Malay <sup>ul</sup>
id	Indonesian <sup>ul</sup>	mt	Maltese
ig	Igbo	mua	Mundang

my	Burmese	sn	Shona
mzn	Mazanderani	so	Somali
naq	Nama	sq	Albanian <sup>ul</sup>
nb	Norwegian Bokmål <sup>ul</sup>	sr-Cyrl-BA	Serbian <sup>ul</sup>
nd	North Ndebele	sr-Cyrl-ME	Serbian <sup>ul</sup>
ne	Nepali	sr-Cyrl-XK	Serbian <sup>ul</sup>
nl	Dutch <sup>ul</sup>	sr-Cyrl	Serbian <sup>ul</sup>
nmg	Kwasio	sr-Latn-BA	Serbian <sup>ul</sup>
nn	Norwegian Nynorsk <sup>ul</sup>	sr-Latn-ME	Serbian <sup>ul</sup>
nnh	Ngiemboon	sr-Latn-XK	Serbian <sup>ul</sup>
nus	Nuer	sr-Latn	Serbian <sup>ul</sup>
nyn	Nyankole	sr	Serbian <sup>ul</sup>
om	Oromo	sv	Swedish <sup>ul</sup>
or	Odia	sw	Swahili
os	Ossetic	ta	Tamil <sup>u</sup>
pa-Arab	Punjabi	te	Telugu <sup>ul</sup>
pa-Guru	Punjabi	teo	Teso
pa	Punjabi	th	Thai <sup>ul</sup>
pl	Polish <sup>ul</sup>	ti	Tigrinya
pms	Piedmontese <sup>ul</sup>	tk	Turkmen <sup>ul</sup>
ps	Pashto	to	Tongan
pt-BR	Portuguese <sup>ul</sup>	tr	Turkish <sup>ul</sup>
pt-PT	Portuguese <sup>ul</sup>	twq	Tasawaq
pt	Portuguese <sup>ul</sup>	tzm	Central Atlas Tamazight
qu	Quechua	ug	Uyghur
rm	Romansh <sup>ul</sup>	uk	Ukrainian <sup>ul</sup>
rn	Rundi	ur	Urdu <sup>ul</sup>
ro	Romanian <sup>ul</sup>	uz-Arab	Uzbek
rof	Rombo	uz-Cyrl	Uzbek
ru	Russian <sup>ul</sup>	uz-Latn	Uzbek
rw	Kinyarwanda	uz	Uzbek
rwk	Rwa	vai-Latn	Vai
sa-Beng	Sanskrit	vai-Vaii	Vai
sa-Deva	Sanskrit	vai	Vai
sa-Gujr	Sanskrit	vi	Vietnamese <sup>ul</sup>
sa-Knda	Sanskrit	vun	Vunjo
sa-Mlym	Sanskrit	wae	Walser
sa-Telu	Sanskrit	xog	Soga
sa	Sanskrit	yav	Yangben
sah	Sakha	yi	Yiddish
saq	Samburu	yo	Yoruba
sbp	Sangu	yue	Cantonese
se	Northern Sami <sup>ul</sup>	zgh	Standard Moroccan Tamazight
seh	Sena		
ses	Koyraboro Senni	zh-Hans-HK	Chinese
sg	Sango	zh-Hans-MO	Chinese
shi-Latn	Tachelhit	zh-Hans-SG	Chinese
shi-Tfng	Tachelhit	zh-Hans	Chinese
shi	Tachelhit	zh-Hant-HK	Chinese
si	Sinhala	zh-Hant-MO	Chinese
sk	Slovak <sup>ul</sup>	zh-Hant	Chinese
sl	Slovenian <sup>ul</sup>	zh	Chinese
smn	Inari Sami	zu	Zulu

---

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`.

---

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

---

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



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

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

northernluri  
northernnsami  
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  
swedish  
swissgerman  
tachelhit-latin  
tachelhit-latn  
tachelhit-tfng  
tachelhit-tifinagh  
tachelhit  
taita  
tamil  
tasawaq  
telugu  
teso  
thai  
tibetan  
tigrinya  
tongan  
turkish  
turkmen  
ukenglish  
ukrainian  
upporsorbian  
urdu

usenglish	vai-vaii
usorbian	vai
uyghur	vietnam
uzbek-arab	vietnamese
uzbek-arabic	vunjo
uzbek-cyrillic	walser
uzbek-cyrl	welsh
uzbek-latin	westernfrisian
uzbek-latn	yangben
uzbek	yiddish
vai-latin	yoruba
vai-latn	zarma
vai-vai	zulu afrikaans

---

### Modifying and adding values to ini files

**New 3.39** There is a way to modify the values of ini files when they get loaded with `\babelprovide` and `import`. To set, say, `digits.native` in the `numbers` section, use something like `numbers/digits.native=abcdefghijklj`. Keys may be added, too. Without `import` you may modify the identification keys.

This can be used to create private variants easily. All you need is to import the same ini file with a different locale name and different parameters.

## 1.14 Selecting fonts

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

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

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

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

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

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

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

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

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

---

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

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 fontspec Warning: 'Language 'LANG' not available for font 'FONT' with script 'SCRIPT' 'Default' language used instead'.*

**This is *not* and error.** This warning is shown by `fontspec`, not by `babel`. It can be irrelevant for English, but not for many other languages, including Urdu and Turkish. This is a useful and harmless warning, and if everything is fine with your document the best thing you can do is just to ignore it altogether.

**TROUBLESHOOTING** *Package babel Info: The following fonts are not babel standard families.*

**This is *not* and 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.

## 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\captionsenglish{%  
  \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. In such a case, the ini file set in the corresponding `babel-<language>.tex` (where `<language>` is the last argument in `\babelprovide`) is imported. See the list of recognized languages above. So, the previous example can be written:

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

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

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

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

**captions=**  $\langle\text{language-tag}\rangle$

Loads only the strings. For example:

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

**hyphenrules=**  $\langle\text{language-list}\rangle$

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

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

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

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

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

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

**New 3.58** Another special value is unhyphenated, which activates a line breking mode that allows spaces to be stretched to arbitrary amounts.

**main** This valueless option makes the language the main one (thus overriding that set when babel is loaded). Only in newly defined languages.

**EXAMPLE** Let's assume your document is mainly in Polytonic Greek, 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]{babel}  
\babelprovide[import, main]{polytonicgreek}
```

Remember there is an alternative syntax for the latter:

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

**script=**  $\langle\text{script-name}\rangle$

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



**language=**  $\langle\text{language-name}\rangle$

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

**alph=**  $\langle\text{counter-name}\rangle$

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

**Alph=**  $\langle\text{counter-name}\rangle$

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`

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

**NOTE** An alternative approach with luatex and Harfbuzz is the `font` option `RawFeature={multiscript=auto}`. It does not switch the babel language and therefore the line breaking rules, but in many cases it can be enough.

**intraspace=**  $\langle\text{base}\rangle$   $\langle\text{shrink}\rangle$   $\langle\text{stretch}\rangle$

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=**  $\langle\text{penalty}\rangle$

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).

**justification=** `kashida` | `elongated` | `unhyphenated`

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

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

**mapfont=** `direction`

Assigns the font for the writing direction of this language (only with `bidi=basic`). Whenever possible, instead of this option use `onchar`, based on the script, which usually

makes more sense. More precisely, what `mapfont=direction` means is, ‘when a character has the same direction as the script for the “provided” language, then change its font to that set for this language’. There are 3 directions, following the bidi Unicode algorithm, namely, Arabic-like, Hebrew-like and left to right. So, there should be at most 3 directives of this kind.

**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} % Telugu better with XeTeX
% Or also, if you want:
% \babelprovide[import, maparabic]{telugu}
\babelfont{rm}{Gautami}
\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.

**New 4.41** Many ‘ini’ locale files has been extended with information about non-positional numerical systems, based on those predefined in CSS. They only work with xetex and luatex and are fully expendable (even inside an unprotected `\edef`). Currently, they are limited to numbers below 10000.

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

- `\localenumerals{<style>}{<number>}`, like `\localenumerals{abjad}{15}`

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

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

The styles are:

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

**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=..>`]{`<year>`}{`<month>`}{`<day>`}

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

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

## 1.19 Accessing language info

**\language** `\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).

`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.

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

**WARNING** **New 3.46** As of version 3.46 `tag.bcp47` returns the full BCP 47 tag. Formerly it returned just the language subtag, which was clearly counterintuitive.

**\getlocaleproperty** `*{\macro}{\locale}{\property}`

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

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

the macro `\hechap` will contain the string פֶּרֶק.

If the key does not exist, the macro is set to `\relax` and an error is raised. **New 3.47** With the starred version no error is raised, so that you can take your own actions with undefined properties.

Babel remembers which ini files have been loaded. There is a loop named

`\LocaleForEach` to traverse the list, where #1 is the name of the current item, so that

`\LocaleForEach{\message{ **#1** }}` just shows the loaded ini's.

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

`\localeid`

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

**NOTE** The `\localeid` is not the same as the `\language` identifier, which refers to a set of hyphenation patterns (which, in turn, is just a component of the line breaking algorithm described in the next section). The data about preloaded patterns are 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 (where it makes sense) as an attribute, too.

## 1.20 Hyphenation and line breaking

Babel deals with three kinds of line breaking rules: Western, typically the LGC group, South East Asian, like Thai, and CJK, but support depends on the engine: `pdfTeX` only deals with the former, `xetex` also with the second one (although in a limited way), while `luatex` provides basic rules for the latter, too.

`\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 `\babenullhyphen`; (3) a break after the hyphen is forbidden if preceded by a

glue >0 pt (at the beginning of a word, provided it is not immediately preceded by, say, a parenthesis).

**\babelhyphenation** [*<language>*, *<language>*, ...]{*<exceptions>*}

**New 3.9a** Sets hyphenation exceptions for the languages given or, without the optional argument, for *all* languages (eg, proper nouns or common loan words, and of course monolingual documents). 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** To set hyphenation exceptions in the preamble before any language is explicitly set with a selector, use \babelhyphenation instead of \hyphenation. 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

<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.

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}
```

Here are the transforms currently predefined. (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.
Greek	<code>diaeresis.hyphen</code>	Removes the diaeresis above iota and upsilon if hyphenated just before. It works with the three variants.
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: <i>!?:;</i> .
Hungarian	<code>digraphs.hyphen</code>	Hyphenates the long digraphs <i>ccs, ddz, ggy, lly, nny, ssz, tty</i> and <i>zsz</i> as <i>cs-cs, dz-dz</i> , etc.

<sup>15</sup>They are similar in concept, but not the same, as those in Unicode.

Indic scripts	danda.nobreak	Prevents a line break before a danda or double danda if there is a space. For Assamese, Bengali, Gujarati, Hindi, Kannada, Malayalam, Marathi, Oriya, Tamil, Telugu.
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.
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.

**\babelposthyphenation**  $\{\langle hyphenrules-name \rangle\}\{\langle lua-pattern \rangle\}\{\langle replacement \rangle\}$

**New 3.37-3.39** With *luatex* it is now 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. Only a few rules are currently provided (see below), 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  $([\text{t}\acute{u}])$ , the replacement could be  $\{1|\text{t}\acute{u}|\text{t}\acute{u}\}$ , which maps  $\text{t}\acute{u}$  to  $\text{t}\acute{u}$ , and  $\acute{u}$  to  $\acute{u}$ , so that the diaeresis is removed.

This feature is activated with the first `\babelposthyphenation` or `\babelprehyphenation`. 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**  $\{\langle locale-name \rangle\}\{\langle lua-pattern \rangle\}\{\langle replacement \rangle\}$

**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. 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  $\text{ž}$  as zh and  $\text{š}$  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{
  autload.bcp47 = on,
  autload.bcp47.options = import
}

\begin{document}

```

```
Chapter in Danish: \chaptername.
```

```
\selectlanguage{de-AT}
```

```
\localedate{2020}{1}{30}
```

```
\end{document}
```

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

`autoload.bcp47` with values on and off.

`autoload.bcp47.options`, which are passed to `\babelprovide`; empty by default, but you may add import (features defined in the corresponding `babel-...tex` file might not be available).

`autoload.bcp47.prefix`. Although the public name used in selectors is the tag, the internal name will be different and generated by prepending a prefix, which by default is `bcp47-`. You may change it with this key.

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

```
\babeladjust{ bcp47.toname = on }
```

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

## 1.23 Selecting scripts

Currently babel provides no standard interface to select scripts, because they are best selected with either `\fontencoding` (low-level) or a language name (high-level). Even the Latin script may require different encodings (ie, sets of glyphs) depending on the language, and therefore such a switch would be in a sense incomplete.<sup>16</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>17</sup>

`\ensureascii`  $\langle text \rangle$

**New 3.9i** This macro makes sure  $\langle text \rangle$  is typeset with a LICR-savvy encoding in the ASCII range. It is used to redefine `\TeX` and `\LaTeX` so that they are correctly typeset even with

<sup>16</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>17</sup>But still defined for backwards compatibility.

LGR or X2 (the complete list is stored in `\BabelNonASCII`, which by default is LGR, X2, OT2, OT3, OT6, LHE, LWN, LMA, LMC, LMS, LMU, but you can modify it). So, in some sense it fixes the bug described in the previous paragraph.

If non-ASCII encodings are not loaded (or no encoding at all), it is no-op (also `\TeX` and `\LaTeX` are not redefined); otherwise, `\ensureascii` switches to the encoding at the beginning of the document if ASCII-savvy, or else the last ASCII-savvy encoding loaded. For example, if you load `LY1`, `LGR`, then it is set to `LY1`, but if you load `LY1`, `T2A` it is set to `T2A`. The symbol encodings `TS1`, `T3`, and `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 is progress in the latter, too, but for example cases may fail).

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

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

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

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

\end{document}
```

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

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

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

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

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

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

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

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

With counters, `\arabic` is not only considered L text always (with `\babelsublr`, see below), but also an “isolated” block which does not interact with the surrounding chars. So, while `1.2` in R text is rendered in that order with `bidi=basic` (as a decimal number), in `\arabic{c1}.\arabic{c2}` the visual order is `c2.c1`. Of course, you may always adjust the order by changing the language, if necessary.<sup>18</sup>

**lists** required in `xetex` and `pdftex`, but only in bidirectional (with both R and L paragraphs) documents in `luatex`.

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

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

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

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

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

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

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

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

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

<sup>18</sup>Next on the roadmap are counters and numeral systems in general. Expect some minor readjustments.

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

**\babelsublr**  $\{\langle lr\text{-}text\rangle\}$

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

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

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

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

**\BabelPatchSection**  $\{\langle section\text{-}name\rangle\}$

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**  $\{\langle cmd\rangle\}\{\langle local\text{-}language\rangle\}\{\langle before\rangle\}\{\langle after\rangle\}$

**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.

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

The same name can be applied to several events. Hooks may be enabled and disabled for all defined events with `\EnableBabelHook{\name}`, `\DisableBabelHook{\name}`.

Names containing the string `babel` are reserved (they are used, for example, by `\usesshortands*` to add a hook for the event `afterextras`). **New 3.33** They may be also applied to a specific language with the optional argument; language-specific settings are executed after global ones.

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

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

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

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

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

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

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

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

**beforeextras** Just before executing `\extras⟨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`.

**\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** uppersorbian  
**Welsh** welsh

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

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

```

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

```

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

---

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

## 1.28 Unicode character properties in luatex

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

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

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

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

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

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

## 1.29 Tweaking some features

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

**New 3.36** Sometimes you might need to disable some babel features. Currently this macro understands the following keys (and only for luatex), with values on or off: `bidi.text`, `bidi.mirroring`, `bidi.mapdigits`, `layout.lists`, `layout.tabular`, `linebreak.sea`, `linebreak.cjk`, `justify.arabic`. For example, you can set `\babeladjust{bidi.text=off}` if you are using an alternative algorithm or with large sections not requiring it. Use with care, because these options do not deactivate other related options (like paragraph direction with `bidi.text`).

## 1.30 Tips, workarounds, known issues and notes

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

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

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

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

```
\addto\extrasfrench{\inputencoding{latin1}}
\addto\extrarussian{\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 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.

<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, `\savingsphcodes` 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.

Also interesting are differences in the sentence structure or related to it. For example, in Basque the number precedes the name (including chapters), in Hungarian “from (1)” is “(1)-ből”, but “from (3)” is “(3)-ből”, in Spanish an item labelled “3.<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 wiki.

#### 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 wiki for further details.

## 2 Loading languages with `language.dat`

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

**New 3.9q** With luatex, however, patterns are loaded on the fly when requested by the language (except the “0th” language, typically english, which is preloaded always).<sup>22</sup> Until 3.9n, this task was delegated to the package `luatex-hyphen`, by Khaled Hosny, Élie Roux, and Manuel Pégourié-Gonnard, and required an extra file named `language.dat.lua`, but now a new mechanism has been devised based solely on `language.dat`. **You must rebuild the formats** if upgrading from a previous version. You may want to have a local `language.dat` for a particular project (for example, a book on Chemistry).<sup>23</sup>

### 2.1 Format

In that file the person who maintains a  $\TeX$  environment has to record for which languages he has hyphenation patterns *and* in which files these are stored<sup>24</sup>. When hyphenation exceptions are stored in a separate file this can be indicated by naming that file *after* the file with the hyphenation patterns.

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

<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.

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

dutch       hyphen.dutch exceptions.dutch % Nederlands
german      hyphen.ger
```

You may also set the font encoding the patterns are intended for by following the language name by a colon and the encoding code.<sup>25</sup> For example:

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

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

A typical error when using babel is the following:

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

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

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

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

The following assumptions are made:

- Some of the language-specific definitions might be used by plain  $\text{T}_{\text{E}}\text{X}$  users, so the files have to be coded so that they can be read by both  $\text{\LaTeX}$  and plain  $\text{T}_{\text{E}}\text{X}$ . The current format can be checked by looking at the value of the macro `\fmtname`.
- The common part of the babel system redefines a number of macros and environments (defined previously in the document style) to put in the names of macros that replace the previously hard-wired texts. These macros have to be defined in the language definition files.
- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are `\<lang>hyphenmins`, `\captions<lang>`, `\date<lang>`, `\extras<lang>` and `\noextras<lang>` (the last two may be left empty); where `<lang>` is either the name of the language definition file or the name of the  $\text{\LaTeX}$  option that is to be used. These macros and their functions are

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

discussed below. You must define all or none for a language (or a dialect); defining, say, `\date⟨lang⟩` but not `\captions⟨lang⟩` does not raise an error but can lead to unexpected results.

- When a language definition file is loaded, it can define `\l@⟨lang⟩` to be a dialect of `\language0` when `\l@⟨lang⟩` is undefined.
- Language names must be all lowercase. If an unknown language is selected, babel will attempt setting it after lowercasing its name.
- The semantics of modifiers is not defined (on purpose). In most cases, they will just be simple separated options (eg, spanish), but a language might require, say, a set of options organized as a tree with suboptions (in such a case, the recommended separator is `/`).

Some recommendations:

- The preferred shorthand is `"`, which is not used in  $\text{\LaTeX}$  (quotes are entered as ``` and `'`). Other good choices are characters which are not used in a certain context (eg, `=` in an ancient language). Note however `=`, `<`, `>`, `:` and the like can be dangerous, because they may be used as part of the syntax of some elements (numeric expressions, key/value pairs, etc.).
- Captions should not contain shorthands or encoding-dependent commands (the latter is not always possible, but should be clearly documented). They should be defined using the LICR. You may also use the new tools for encoded strings, described below.
- Avoid adding things to `\noextras⟨lang⟩` except for `umlauthigh` and friends, `\bbl@deactivate`, `\bbl@(non) frenchspacing`, and language-specific macros. Use always, if possible, `\bbl@save` and `\bbl@savevariable` (except if you still want to have access to the previous value). Do not reset a macro or a setting to a hardcoded value. Never. Instead save its value in `\extras⟨lang⟩`.
- Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the font encoding (low-level) or the language (high-level, which in turn may switch the font encoding). Usage of things like `\latintext` is deprecated.<sup>26</sup>
- Please, for “private” internal macros do not use the `\bbl@` prefix. It is used by babel and it can lead to incompatibilities.

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

### 3.1 Guidelines for contributed languages

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

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

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

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

- 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, ot f, mf files and the like, but also fd ones.
- Font and input encodings are usually best placed in the corresponding tree, too, but sometimes they belong more naturally to the babel style. Note you may also need to define a LICR.
- Babel ldf files may just interface a framework, as it happens often with Oriental languages/scripts. This framework is best placed in its own directory.

The following page provides a starting point for ldf files:

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

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

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

## 3.2 Basic macros

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

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

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

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

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

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

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

**\captions<lang>** The macro \captions<lang> defines the macros that hold the texts to replace the original hard-wired texts.

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

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

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



<code>\bbl@declare@ttribute</code>	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.
<code>\main@language</code>	To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use <code>\main@language</code> instead of <code>\selectlanguage</code> . This will just store the name of the language, and the proper language will be activated at the start of the document.
<code>\ProvidesLanguage</code>	The macro <code>\ProvidesLanguage</code> should be used to identify the language definition files. Its syntax is similar to the syntax of the $\TeX$ command <code>\ProvidesPackage</code> .
<code>\LdfInit</code>	The macro <code>\LdfInit</code> 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 <code>@</code> -sign, preventing the <code>.ldf</code> file from being processed twice, etc.
<code>\ldf@quit</code>	The macro <code>\ldf@quit</code> does work needed if a <code>.ldf</code> file was processed earlier. This includes resetting the category code of the <code>@</code> -sign, preparing the language to be activated at <code>\begin{document}</code> time, and ending the input stream.
<code>\ldf@finish</code>	The macro <code>\ldf@finish</code> does work needed at the end of each <code>.ldf</code> file. This includes resetting the category code of the <code>@</code> -sign, loading a local configuration file, and preparing the language to be activated at <code>\begin{document}</code> time.
<code>\loadlocalcfg</code>	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 <code>\captions{&lt;lang&gt;}</code> to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by <code>\ldf@finish</code> .
<code>\substitutefontfamily</code>	(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 <code>.fd</code> file will instruct $\TeX$ to use a font from the second family when a font from the first family in the given encoding seems to be needed.

### 3.3 Skeleton

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

```

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

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

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

\bbl@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\monthiname{<name of first month>}
% More strings

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

\StartBabelCommands*{<dialect>}{date}
\SetString\monthiname{<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`. 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`  
`\bbl@nonfrenchspacing` The commands `\bbl@frenchspacing` and `\bbl@nonfrenchspacing` can be used to properly switch French spacing on and off.

### 3.8 Encoding-dependent strings

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

Furthermore, it must be activated explicitly, with the package option `strings`. If there is no `strings`, these blocks are ignored, except `\SetCases` (and except if forced as described

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

below). In other words, the old way of defining/switching strings still works and it's used by default.

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

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

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

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

Encoding info is `charset=` followed by a charset, which if given sets how the strings should be translated to the internal representation used by the engine, typically `utf8`, which is the only value supported currently (default is no translations). Note `charset` is applied by luatex and xetex when reading the file, not when the macro or string is used in the document.

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

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

The  $\langle category \rangle$  is either `captions`, `date` or `extras`. You must stick to these three categories, even if no error is raised when using other name.<sup>28</sup> It may be empty, too, but in such a case using `\SetString` is an error (but not `\SetCase`).

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

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

\EndBabelCommands
```

A real example is:

---

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

```

\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\monthiiiname{März}

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

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


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

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

\EndBabelCommands

```

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

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

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>

$\backslash EndBabelCommands$  Marks the end of the series of blocks.

$\backslash AfterBabelCommands$   $\{ \langle code \rangle \}$

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

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

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

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

Use this command to define strings, without including any “logic” if possible, which should be a separated macro. See the example above for the date.

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

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

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

#1 is replaced by the roman numeral.

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

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

```
\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=`I\relax
 \uccode`1=`I\relax}
{\lccode`I=`i\relax
 \lccode`I=`1\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** {*<to-lower-macros>*}

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

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

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

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

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

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

## 4 Changes

### 4.1 Changes in babel version 3.9

Most of the changes in version 3.9 were related to bugs, either to fix them (there were lots), or to provide some alternatives. Even new features like `\babelhyphen` are intended to solve a certain problem (in this case, the lacking of a uniform syntax and behavior for shorthands across languages). These changes are described in this manual in the corresponding place. A selective list follows:

- `\select@language` did not set `\language`. This meant the language in force when auxiliary files were loaded was the one used in, for example, shorthands – if the language was `german`, a `\select@language{spanish}` had no effect.
- `\foreignlanguage` and `otherlanguage*` messed up `\extras<language>`. Scripts, encodings and many other things were not switched correctly.
- The `:ENC` mechanism for hyphenation patterns used the encoding of the *previous* language, not that of the language being selected.
- `'` (with `activeacute`) had the original value when writing to an auxiliary file, and things like an infinite loop can happen. It worked incorrectly with `^` (if activated) and also if deactivated.
- Active chars were not reset at the end of language options, and that led to incompatibilities between languages.
- `\textormath` raised an error with a conditional.
- `\aliasshorthand` didn't work (or only in a few and very specific cases).
- `\l@english` was defined incorrectly (using `\let` instead of `\chardef`).
- `ldf` files not bundled with `babel` were not recognized when called as global options.

## 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>).

## 5 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.

## 6 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 counter s has been devised to have arbitrary keys, so you can add lowercased keys if you want.

## 7 Tools

```
1 <<version=3.60.2416>>
2 <<date=2021/06/27>>
```

**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@csarg#1#2{\expandafter#1\csname bbl@#2\endcsname}%
12 \def\bbl@cs#1{\csname bbl@#1\endcsname}
13 \def\bbl@cl#1{\csname bbl@#1\language\endcsname}
14 \def\bbl@loop#1#2#3{\bbl@loop#1{#3}#2,\@nnil,}
15 \def\bbl@loopx#1#2{\expandafter\bbl@loop\expandafter#1\expandafter{#2}}
16 \def\bbl@loop#1#2#3,{%
17   \ifx\@nnil#3\relax\else
18     \def#1{#3}#2\bbl@afterfi\bbl@loop#1{#2}%
19   \fi}
20 \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.

```
21 \def\bbl@add@list#1#2{%
22   \edef#1{%
23     \bbl@ifunset{\bbl@stripslash#1}%
24     {}%
25     {\ifx#1\@empty\else#1,\fi}%
26   #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.

```
27 \long\def\bbl@afterelse#1\else#2\fi{\fi#1}
28 \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` and `\<.>` for `\noexpand` applied to a built macro name (the latter does not define the macro if undefined to `\relax`, because it is created locally). The result may be followed by extra arguments, if necessary.

```
29 \def\bbl@exp#1{%
30   \begingroup
31   \let\ \noexpand
32   \def\<##1>{\expandafter\noexpand\csname##1\endcsname}%
33   \edef\bbl@exp@aux{\endgroup#1}%
34   \bbl@exp@aux}
```

<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.



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

```

35 \def\bbl@tempa#1{%
36   \long\def\bbl@trim##1##2{%
37     \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
38   \def\bbl@trim@c{%
39     \ifx\bbl@trim@a\@sptoken
40       \expandafter\bbl@trim@b
41     \else
42       \expandafter\bbl@trim@b\expandafter#1%
43     \fi}%
44   \long\def\bbl@trim@b#1##1 \@nil{\bbl@trim@i##1}}
45 \bbl@tempa{ }
46 \long\def\bbl@trim@i#1\@nil#2\relax#3{##3{##1}}
47 \long\def\bbl@trim@def#1{\bbl@trim{\def#1}}

```

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

```

48 \begingroup
49   \gdef\bbl@ifunset#1{%
50     \expandafter\ifx\csname#1\endcsname\relax
51       \expandafter\@firstoftwo
52     \else
53       \expandafter\@secondoftwo
54     \fi}
55 \bbl@ifunset{ifcsname}%
56 {}%
57 {\gdef\bbl@ifunset#1{%
58   \ifcsname#1\endcsname
59     \expandafter\ifx\csname#1\endcsname\relax
60       \bbl@afterelse\expandafter\@firstoftwo
61     \else
62       \bbl@afterfi\expandafter\@secondoftwo
63     \fi
64   \else
65     \expandafter\@firstoftwo
66   \fi}}
67 \endgroup

```

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

```

68 \def\bbl@ifblank#1{%
69   \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
70 \long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
71 \def\bbl@ifset#1#2#3{%
72   \bbl@ifunset{##1}{##3}{\bbl@exp{\bbl@ifblank{##1}}{##3}{##2}}}

```

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

```

73 \def\bbl@forkv#1#2{%
74   \def\bbl@kvcmd##1##2##3{##2}%
75   \bbl@kvnext#1,\@nil,}
76 \def\bbl@kvnext#1,{%

```

```

77 \ifx\@nil#1\relax\else
78 \bbl@ifblank{#1}{\bbl@forkv@eq#1=@empty=@nil{#1}}%
79 \expandafter\bbl@kvnext
80 \fi}
81 \def\bbl@forkv@eq#1=#2=#3\@nil#4{%
82 \bbl@trim@def\bbl@forkv@a{#1}%
83 \bbl@trim{\expandafter\bbl@kvcmd\expandafter{\bbl@forkv@a}}{#2}{#4}}

```

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

```

84 \def\bbl@vforeach#1#2{%
85 \def\bbl@forcmd##1{#2}%
86 \bbl@fornext#1,\@nil,}
87 \def\bbl@fornext#1,{%
88 \ifx\@nil#1\relax\else
89 \bbl@ifblank{#1}{\bbl@trim\bbl@forcmd{#1}}%
90 \expandafter\bbl@fornext
91 \fi}
92 \def\bbl@foreach#1{\expandafter\bbl@vforeach\expandafter{#1}}

```

\bbl@replace

```

93 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
94 \toks@{}}%
95 \def\bbl@replace@aux##1#2##2#2{%
96 \ifx\bbl@nil##2%
97 \toks@\expandafter{\the\toks@##1}%
98 \else
99 \toks@\expandafter{\the\toks@##1#3}%
100 \bbl@afterfi
101 \bbl@replace@aux##2#2%
102 \fi}%
103 \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
104 \edef#1{\the\toks@}}

```

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

```

105 \ifx\detokenize\undefined\else % Unused macros if old Plain TeX
106 \bbl@exp{\def\\bbl@parsedef##1\detokenize{macro:}}#2->#3\relax{%
107 \def\bbl@tempa{#1}%
108 \def\bbl@tempb{#2}%
109 \def\bbl@tempe{#3}}
110 \def\bbl@sreplace#1#2#3{%
111 \begingroup
112 \expandafter\bbl@parsedef\meaning#1\relax
113 \def\bbl@tempc{#2}%
114 \def\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
115 \def\bbl@tempd{#3}%
116 \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
117 \bbl@xin@{\bbl@tempc}{\bbl@tempe}% If not in macro, do nothing
118 \ifin@
119 \bbl@exp{\\bbl@replace\\bbl@tempe{\bbl@tempc}{\bbl@tempd}}%
120 \def\bbl@tempc{% Expanded an executed below as 'uplevel'
121 \\\makeatletter % "internal" macros with @ are assumed
122 \\\scantokens{%
123 \bbl@tempa\\@namedef{\bbl@stripslash#1}\bbl@tempb{\bbl@tempe}}%
124 \catcode64=\the\catcode64\relax}% Restore @

```

```

125     \else
126       \let\bbl@tempc\@empty % Not \relax
127     \fi
128     \bbl@exp{%      For the 'uplevel' assignments
129   \endgroup
130     \bbl@tempc}} % empty or expand to set #1 with changes
131 \fi

```

Two further tools. `\bbl@samestring` 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 $\TeX$ , 1 is luatex, and 2 is xetex. You may use the latter in your language style if you want.

```

132 \def\bbl@ifsamestring#1#2{%
133   \begingroup
134     \protected@edef\bbl@tempb{#1}%
135     \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
136     \protected@edef\bbl@tempc{#2}%
137     \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
138     \ifx\bbl@tempb\bbl@tempc
139       \aftergroup\@firstoftwo
140     \else
141       \aftergroup\@secondoftwo
142     \fi
143   \endgroup}
144 \chardef\bbl@engine=%
145 \ifx\directlua\@undefined
146   \ifx\XeTeXinputencoding\@undefined
147     \z@
148   \else
149     \tw@
150   \fi
151 \else
152   \@ne
153 \fi

```

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

```

154 \def\bbl@bspack{%
155   \ifhmode
156     \hskip\z@skip
157     \def\bbl@espack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
158   \else
159     \let\bbl@espack\@empty
160   \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`.

```

161 \def\bbl@cased{%
162   \ifx\oe\OE
163     \expandafter\in@\expandafter
164       {\expandafter\OE\expandafter}\expandafter{\oe}%
165     \ifin@
166       \bbl@afterelse\expandafter\MakeUppercase
167     \else
168       \bbl@afterfi\expandafter\MakeLowercase
169     \fi
170   \else
171     \expandafter\@firstofone
172   \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.

```

173 \def\bbl@extras@wrap#1#2#3{% 1:in-test, 2:before, 3:after
174   \toks@{\expandafter\expandafter\expandafter{%
175     \csname extras\language\endcsname}%
176     \bbl@exp{\in@{#1}{\the\toks@}}}%
177   \ifin@ \else
178     \@temptokena{#2}%
179     \edef\bbl@tempc{\the\@temptokena\the\toks@}%
180     \toks@\expandafter{\bbl@tempc#3}%
181     \expandafter\edef\csname extras\language\endcsname{\the\toks@}%
182   \fi}
183 <</Basic macros>>

```

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

```

184 <<*Make sure ProvidesFile is defined>> ≡
185 \ifx\ProvidesFile\@undefined
186   \def\ProvidesFile#1[#2 #3 #4]{%
187     \wlog{File: #1 #4 #3 <#2>}%
188     \let\ProvidesFile\@undefined}
189 \fi
190 <</Make sure ProvidesFile is defined>>

```

## 7.1 Multiple languages

**\language** 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. 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.

```

191 <<*Define core switching macros>> ≡
192 \ifx\language\@undefined
193   \csname newcount\endcsname\language
194 \fi
195 <</Define core switching macros>>

```

**\last@language** Another counter is used to store the last language defined. For pre-3.0 formats an extra counter has to be allocated.

**\addlanguage** This macro was introduced for  $\text{\TeX}$  < 2. Preserved for compatibility.

```

196 <<*Define core switching macros>> ≡
197 <<*Define core switching macros>> ≡
198 \countdef\last@language=19 % TODO. why? remove?
199 \def\addlanguage{\csname newlanguage\endcsname}
200 <</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 or  $\text{\LaTeX}$  2.09. 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.

## 7.2 The Package File ( $\text{\LaTeX}$ , `babel.sty`)

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

Many of the following options don't do anything themselves, they are just defined in order to make it possible for babel and language definition files to check if one of them was specified by the user. The first two options are for debugging.

```

201 (*package)
202 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
203 \ProvidesPackage{babel}[\langle\date\rangle\langle\version\rangle] The Babel package]
204 \@ifpackagewith{babel}{debug}
205   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}}%
206   \let\bbl@debug\@firstofone
207   \ifx\directlua\@undefined\else
208     \directlua{ Babel = Babel or {}
209       Babel.debug = true }%
210   \fi}
211 {\providecommand\bbl@trace[1]{}}%
212 \let\bbl@debug\@gobble
213 \ifx\directlua\@undefined\else
214   \directlua{ Babel = Babel or {}
215     Babel.debug = false }%
216   \fi}
217 \langle\Basic macros\rangle
218 % Temporarily repeat here the code for errors. TODO.
219 \def\bbl@error#1#2{%
220   \begingroup
221     \def\{\MessageBreak}%
222     \PackageError{babel}{#1}{#2}%
223   \endgroup}
224 \def\bbl@warning#1{%
225   \begingroup
226     \def\{\MessageBreak}%
227     \PackageWarning{babel}{#1}%
228   \endgroup}
229 \def\bbl@infowarn#1{%
230   \begingroup
231     \def\{\MessageBreak}%
232     \GenericWarning
233       {(babel) \@spaces\@spaces\@spaces}%
234       {Package babel Info: #1}%
235   \endgroup}
236 \def\bbl@info#1{%
237   \begingroup
238     \def\{\MessageBreak}%
239     \PackageInfo{babel}{#1}%
240   \endgroup}
241 \def\bbl@nocaption{\protect\bbl@nocaption@i}
242 % TODO - Wrong for \today !!! Must be a separate macro.
243 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
244   \global\@namedef{#2}{\textbf{?#1?}}%
245   \@nameuse{#2}%
246   \edef\bbl@tempa{#1}%
247   \bbl@sreplace\bbl@tempa{name}{}}%
248   \bbl@warning{%
249     \@backslashchar#1 not set for '\language' name'. Please,\%
250     define it after the language has been loaded\%
251     (typically in the preamble) with\%
252     \string\setlocalecaption{\language name}{\bbl@tempa}{..}\%
253     Reported}}
254 \def\bbl@tentative{\protect\bbl@tentative@i}
255 \def\bbl@tentative@i#1{%

```

```

256 \bbl@warning{%
257   Some functions for '#1' are tentative.\\%
258   They might not work as expected and their behavior\\%
259   may change in the future.\\%
260   Reported}}
261 \def\nolanerr#1{%
262   \bbl@error
263   {You haven't defined the language '#1' yet.\\%
264     Perhaps you misspelled it or your installation\\%
265     is not complete}%
266   {Your command will be ignored, type <return> to proceed}}
267 \def\nopatterns#1{%
268   \bbl@warning
269   {No hyphenation patterns were preloaded for\\%
270     the language '#1' into the format.\\%
271     Please, configure your TeX system to add them and\\%
272     rebuild the format. Now I will use the patterns\\%
273     preloaded for \bbl@nulllanguage\space instead}}
274   % End of errors
275 \@ifpackagewith{babel}{silent}
276   {\let\bbl@info@gobble
277    \let\bbl@infowarn@gobble
278    \let\bbl@warning@gobble}
279   {}
280 %
281 \def\AfterBabelLanguage#1{%
282   \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.

```

283 \ifx\bbl@languages\undefined\else
284   \begingroup
285     \catcode\^^I=12
286     \@ifpackagewith{babel}{showlanguages}{%
287       \begingroup
288         \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
289         \wlog{<*languages>}%
290         \bbl@languages
291         \wlog{</languages>}%
292       \endgroup}{%
293     \endgroup
294     \def\bbl@elt#1#2#3#4{%
295       \ifnum#2=\z@
296         \gdef\bbl@nulllanguage{#1}%
297         \def\bbl@elt##1##2##3##4{}%
298       \fi}%
299     \bbl@languages
300 \fi%

```

### 7.3 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`.

```

301 \bbl@trace{Defining option 'base'}
302 \@ifpackagewith{babel}{base}{%

```

```

303 \let\bbl@onlyswitch\@empty
304 \let\bbl@provide@locale\relax
305 \input babel.def
306 \let\bbl@onlyswitch\@undefined
307 \ifx\directlua\@undefined
308   \DeclareOption*{\bbl@patterns{\CurrentOption}}%
309 \else
310   \input luababel.def
311   \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
312 \fi
313 \DeclareOption{base}{}%
314 \DeclareOption{showlanguages}{}%
315 \ProcessOptions
316 \global\expandafter\let\csname opt@babel.sty\endcsname\relax
317 \global\expandafter\let\csname ver@babel.sty\endcsname\relax
318 \global\let\@ifl@ter@\@ifl@ter
319 \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
320 \endinput{}%
321% \end{macrocode}
322%
323% \subsection{\texttt{key=value} options and other general option}
324%
325%   The following macros extract language modifiers, and only real
326%   package options are kept in the option list. Modifiers are saved
327%   and assigned to |\BabelModifiers| at |\bbl@load@language|; when
328%   no modifiers have been given, the former is |\relax|. How
329%   modifiers are handled are left to language styles; they can use
330%   |\in@|, loop them with |\@for| or load |keyval|, for example.
331%
332%   \begin{macrocode}
333\bbl@trace{key=value and another general options}
334\bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
335\def\bbl@tempb#1.#2{% Remove trailing dot
336  #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
337\def\bbl@tempd#1.#2\@nnil{% TODO. Refactor lists?
338  \ifx\@empty#2%
339    \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
340  \else
341    \in@{,provide=}{, #1}%
342    \ifin@
343      \edef\bbl@tempc{%
344        \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
345    \else
346      \in@{=}{ #1}%
347      \ifin@
348        \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
349      \else
350        \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
351        \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
352      \fi
353    \fi
354  \fi}
355\let\bbl@tempc\@empty
356\bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
357\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.

```

358 \DeclareOption{KeepShorthandsActive}{}
359 \DeclareOption{activeacute}{}
360 \DeclareOption{activegrave}{}
361 \DeclareOption{debug}{}
362 \DeclareOption{noconfigs}{}
363 \DeclareOption{showlanguages}{}
364 \DeclareOption{silent}{}
365 % \DeclareOption{mono}{}
366 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
367 \chardef\bbl@iniflag\z@
368 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main -> +1
369 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@} % add = 2
370 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % add + main
371 % A separate option
372 \let\bbl@autoload@options\@empty
373 \DeclareOption{provide@=*}{\def\bbl@autoload@options{import}}
374 % Don't use. Experimental. TODO.
375 \newif\ifbbl@single
376 \DeclareOption{selectors=off}{\bbl@singletrue}
377 <<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.

```

378 \let\bbl@opt@shorthands\@nnil
379 \let\bbl@opt@config\@nnil
380 \let\bbl@opt@main\@nnil
381 \let\bbl@opt@headfoot\@nnil
382 \let\bbl@opt@layout\@nnil
383 \let\bbl@opt@provide\@nnil

```

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

```

384 \def\bbl@tempa#1=#2\bbl@tempa{%
385   \bbl@csarg\ifx{opt@#1}\@nnil
386     \bbl@csarg\edef{opt@#1}{#2}%
387   \else
388     \bbl@error
389     {Bad option '#1=#2'. Either you have misspelled the\\%
390     key or there is a previous setting of '#1'. Valid\\%
391     keys are, among others, 'shorthands', 'main', 'bidi',\\%
392     'strings', 'config', 'headfoot', 'safe', 'math'.}%
393     {See the manual for further details.}
394   \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.

```

395 \let\bbl@language@opts\@empty
396 \DeclareOption*{%
397   \bbl@xin@{\string=}{\CurrentOption}%
398   \ifin@
399     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
400   \else
401     \bbl@add@list\bbl@language@opts{\CurrentOption}%
402   \fi}

```

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

```

403 \ProcessOptions*

```



```

404 \ifx\bbbl@opt@provide\@nnil\else % Tests. Ignore.
405   \chardef\bbbl@iniflag\@ne
406   \bbbl@replace\bbbl@opt@provide{;}{,}
407   \bbbl@add\bbbl@opt@provide{,import}
408   \show\bbbl@opt@provide
409 \fi
410 %

```

## 7.4 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=...

```

411 \bbbl@trace{Conditional loading of shorthands}
412 \def\bbbl@sh@string#1{%
413   \ifx#1\@empty\else
414     \ifx#1t\string~%
415     \else\ifx#1c\string,%
416     \else\string#1%
417     \fi\fi
418   \expandafter\bbbl@sh@string
419   \fi}
420 \ifx\bbbl@opt@shorthands\@nnil
421   \def\bbbl@ifshorthand#1#2#3{#2}%
422 \else\ifx\bbbl@opt@shorthands\@empty
423   \def\bbbl@ifshorthand#1#2#3{#3}%
424 \else

```

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

```

425   \def\bbbl@ifshorthand#1{%
426     \bbbl@xin@\string#1}{\bbbl@opt@shorthands}%
427     \ifin@
428     \expandafter\@firstoftwo
429     \else
430     \expandafter\@secondoftwo
431     \fi}

```

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

```

432   \edef\bbbl@opt@shorthands{%
433     \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.

```

434   \bbbl@ifshorthand{'}%
435   {\PassOptionsToPackage{activeacute}{babel}}{}
436   \bbbl@ifshorthand{`}%
437   {\PassOptionsToPackage{activegrave}{babel}}{}
438 \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.

```

439 \ifx\bbbl@opt@headfoot\@nnil\else
440   \g@addto@macro\@resetactivechars{%
441     \set@typeset@protect
442     \expandafter\select@language@x\expandafter{\bbbl@opt@headfoot}%
443     \let\protect\noexpand}
444 \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 set.

```

445 \ifx\bbl@opt@safe\undefined
446   \def\bbl@opt@safe{BR}
447 \fi
448 \ifx\bbl@opt@main\@nnil\else
449   \edef\bbl@language@opts{%
450     \ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
451     \bbl@opt@main}
452 \fi

```

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

```

453 \bbl@trace{Defining IfBabelLayout}
454 \ifx\bbl@opt@layout\@nnil
455   \newcommand\IfBabelLayout[3]{#3}%
456 \else
457   \newcommand\IfBabelLayout[1]{%
458     \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
459     \ifin@
460       \expandafter\@firstoftwo
461     \else
462       \expandafter\@secondoftwo
463     \fi}
464 \fi

```

**Common definitions.** *In progress.* Still based on `babel.def`, but the code should be moved here.

```

465 \input babel.def

```

## 7.5 Cross referencing macros

The  $\TeX$  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.

```

466 <<More package options>> ≡
467 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
468 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
469 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
470 <</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.

```

471 \bbl@trace{Cross referencing macros}
472 \ifx\bbl@opt@safe\@empty\else
473   \def\@newl@bel#1#2#3{%
474     {\@safe@activestrue
475       \bbl@ifunset{#1@#2}%
476       \relax
477       {\gdef\@multiplelabels{%
478         \latex@warning@no@line{There were multiply-defined labels}}}%

```

```

479      \@latex@warning@no@line{Label `#2' multiply defined}}}%
480      \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.

481 \CheckCommand*\@testdef[3]{%
482   \def\reserved@a{#3}%
483   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
484   \else
485     \@tempswattrue
486   \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.

487 \def\@testdef#1#2#3{% TODO. With @samestring?
488   \@safe@activetrue
489   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
490   \def\bbl@tempb{#3}%
491   \@safe@activetrue
492   \ifx\bbl@tempa\relax
493   \else
494     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
495   \fi
496   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
497   \ifx\bbl@tempa\bbl@tempb
498   \else
499     \@tempswattrue
500   \fi}
501 \fi

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

502 \bbl@xin@{R}\bbl@opt@safe
503 \ifin@
504   \bbl@redefineroast\ref#1{%
505     \@safe@activetrue\org@ref{#1}\@safe@activetrue}
506   \bbl@redefineroast\pageref#1{%
507     \@safe@activetrue\org@pageref{#1}\@safe@activetrue}
508 \else
509   \let\org@ref\ref
510   \let\org@pageref\pageref
511 \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.

512 \bbl@xin@{B}\bbl@opt@safe
513 \ifin@
514   \bbl@redefine\@citex[#1]#2{%
515     \@safe@activetrue\edef\@tempa{#2}\@safe@activetrue
516     \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.

```
517 \AtBeginDocument{%
518   \@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.)

```
519   \def\@citex[#1][#2]#3{%
520     \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
521     \org@citex[#1][#2]{\@tempa}}%
522   }{}}
```

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

```
523 \AtBeginDocument{%
524   \@ifpackageloaded{cite}{%
525     \def\@citex[#1]#2{%
526       \@safe@activetrue\org@citex[#1]{#2}\@safe@activesfalse}%
527     }{}}
```

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

```
528 \bbl@redefine\nocite#1{%
529   \@safe@activetrue\org@nocite{#1}\@safe@activesfalse}
```

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

```
530 \bbl@redefine\bibcite{%
531   \bbl@cite@choice
532   \bibcite}
```

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

```
533 \def\bbl@bibcite#1#2{%
534   \org@bibcite{#1}{\@safe@activesfalse#2}}
```

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

```
535 \def\bbl@cite@choice{%
536   \global\let\bibcite\bbl@bibcite
537   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}}%
538   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}}%
539   \global\let\bbl@cite@choice\relax}
```

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

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

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

```

541 \bbl@redefine\@bibitem#1{%
542   \@safe@activetrue\org@bibitem{#1}\@safe@activesfalse}
543 \else
544   \let\org@nocite\nocite
545   \let\org@@citex\@citex
546   \let\org@bibcite\bibcite
547   \let\org@bibitem\@bibitem
548 \fi

```

## 7.6 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.

```

549 \bbl@trace{Marks}
550 \IfBabelLayout{sectioning}
551   {\ifx\bbl@opt@headfoot\@nnil
552     \g@addto@macro\resetactivechars{%
553       \set@typeset@protect
554       \expandafter\select@language@x\expandafter{\bbl@main@language}%
555       \let\protect\noexpand
556       \ifcase\bbl@bidimode\else % Only with bidi. See also above
557         \edef\thepage{%
558           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
559       \fi}%
560   \fi}
561 {\ifbbl@single\else
562   \bbl@ifunset{markright} \bbl@redefine\bbl@redefineroobust
563   \markright#1{%
564     \bbl@ifblank{#1}%
565     {\org@markright{}}%
566     {\toks@{#1}%
567       \bbl@exp{%
568         \\org@markright{\\protect\\foreignlanguage{\language}%
569           {\\\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.)

`\@mkboth`

```

570   \ifx\@mkboth\markboth
571     \def\bbl@tempc{\let\@mkboth\markboth}
572   \else
573     \def\bbl@tempc{}
574   \fi
575   \bbl@ifunset{markboth} \bbl@redefine\bbl@redefineroobust
576   \markboth#1#2{%
577     \protected@edef\bbl@tempb##1{%
578       \protect\foreignlanguage
579       {\language}{\protect\bbl@restore@actives##1}%
580     \bbl@ifblank{#1}%
581     {\toks@{}}%

```

```

582      {\toks@expandafter{\bbl@tempb{#1}}}%
583      \bbl@ifblank{#2}%
584      {\@temptokena{}}%
585      {\@temptokena\expandafter{\bbl@tempb{#2}}}%
586      \bbl@exp{\org@markboth{\the\toks@}{\the\@temptokena}}%
587      \bbl@tempc
588      \fi} % end ifbbl@single, end \IfBabelLayout

```

## 7.7 Preventing clashes with other packages

### 7.7.1 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.

```

589 \bbl@trace{Preventing clashes with other packages}
590 \bbl@xin@{R}\bbl@opt@safe
591 \ifin@
592   \AtBeginDocument{%
593     \@ifpackageloaded{ifthen}{%
594       \bbl@redefine@long\ifthenelse#1#2#3{%
595         \let\bbl@temp@pref\pageref
596         \let\pageref\org@pageref
597         \let\bbl@temp@ref\ref
598         \let\ref\org@ref
599         \@safe@activestrue
600         \org@ifthenelse{#1}%
601         {\let\pageref\bbl@temp@pref
602          \let\ref\bbl@temp@ref
603          \@safe@activesfalse
604          #2}%
605         {\let\pageref\bbl@temp@pref
606          \let\ref\bbl@temp@ref
607          \@safe@activesfalse
608          #3}%
609       }%
610     }{}%
611   }

```

### 7.7.2 varioref

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

```

612   \AtBeginDocument{%
613     \@ifpackageloaded{varioref}{%

```

```

614 \bbl@redefine\@@vpageref#1[#2]#3{%
615 \@safe@activestrue
616 \org@@vpageref{#1}[#2]#3}%
617 \@safe@activesfalse}%
618 \bbl@redefine\hrefpagenum#1#2{%
619 \@safe@activestrue
620 \org@hrefpagenum{#1}#2}%
621 \@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.

```

622 \expandafter\def\csname Ref \endcsname#1{%
623 \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
624 }{}%
625 }
626 \fi

```

### 7.7.3 `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.

```

627 \AtEndOfPackage{%
628 \AtBeginDocument{%
629 \ifpackageloaded{hhline}%
630 {\expandafter\ifx\csname normal@char\string:\endcsname\relax
631 \else
632 \makeatletter
633 \def\@currname{hhline}\input{hhline.sty}\makeatother
634 \fi}%
635 {}}}

```

### 7.7.4 `hyperref`

`\pdfstringdefDisableCommands` A number of interworking problems between `babel` and `hyperref` are tackled by `hyperref` itself. The following code was introduced to prevent some annoying warnings but it broke bookmarks. This was quickly fixed in `hyperref`, which essentially made it no-op. However, it will not be removed for the moment because `hyperref` is expecting it. TODO. Still true? Commented out in 2020/07/27.

```

636 % \AtBeginDocument{%
637 % \ifx\pdfstringdefDisableCommands\@undefined\else
638 % \pdfstringdefDisableCommands{\languageshorthands{system}}%
639 % \fi}

```

### 7.7.5 `fancyhdr`

`\FOREIGNLANGUAGE` The package `fancyhdr` treats the running head and foot lines somewhat differently as the standard classes. A symptom of this is that the command `\foreignlanguage` which `babel` adds to the marks can end up inside the argument of `\MakeUppercase`. To prevent unexpected results we need to define `\FOREIGNLANGUAGE` here.

```

640 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
641 \lowercase{\foreignlanguage{#1}}}

```

`\substitutefontfamily` 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. This command is deprecated. Use the tools provides by  $\TeX$ .

```

642 \def\substitutefontfamily#1#2#3{%
643   \lowercase{\immediate\openout15=#1#2.fd\relax}%
644   \immediate\write15{%
645     \string\ProvidesFile{#1#2.fd}%
646     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
647     \space generated font description file]^J
648     \string\DeclareFontFamily{#1}{#2}{^^J
649     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
650     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
651     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
652     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
653     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
654     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
655     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
656     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
657   }%
658   \closeout15
659 }
660 \@onlypreamble\substitutefontfamily

```

## 7.8 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of  $\TeX$  and  $\LaTeX$  always come out in the right encoding. There is a list of non-ASCII encodings. Unfortunately, `fontenc` deletes its package options, so we must guess which encodings has been loaded by traversing `\@filelist` to search for `<enc>enc.def`. 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`

```

661 \bbl@trace{Encoding and fonts}
662 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,PU,PD1}
663 \newcommand\BabelNonText{TS1,T3,TS3}
664 \let\org@TeX\TeX
665 \let\org@LaTeX\LaTeX
666 \let\ensureascii\@firstofone
667 \AtBeginDocument{%
668   \in@false
669   \bbl@foreach\BabelNonASCII{% is there a text non-ascii enc?
670     \ifin@false
671       \lowercase{\bbl@xin@{,#1enc.def},{,\@filelist,}}%
672       \fi}%
673   \ifin@ % if a text non-ascii has been loaded
674     \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
675     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
676     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
677     \def\bbl@tempb#1@@{\uppercase{\bbl@tempc#1}ENC.DEF\@empty\@}%
678     \def\bbl@tempc#1ENC.DEF#2\@{\%
679       \ifx\@empty#2\else
680         \bbl@ifunset{T#1}%
681         {}%
682         {\bbl@xin@{,#1,},{,\BabelNonASCII,\BabelNonText,}}%
683         \ifin@
684           \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
685           \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%

```



```

686         \else
687         \def\ensureascii#1{{\fontencoding{#1}\selectfont#1}}%
688         \fi}%
689     \fi}%
690     \bbl@foreach\@filelist{\bbl@tempb#1@@}% TODO - @@ de mas??
691     \bbl@xin@{\,cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
692     \ifin@ \else
693     \edef\ensureascii#1{{%
694         \noexpand\fontencoding{cf@encoding}\noexpand\selectfont#1}}%
695     \fi
696     \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.

```

697 \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.

```

698 \AtBeginDocument{%
699     \@ifpackageloaded{fontspec}%
700     {\xdef\latinencoding{%
701         \ifx\UTFencname\@undefined
702         EU\ifcase\bbl@engine\or2\or1\fi
703         \else
704         \UTFencname
705         \fi}}%
706     {\gdef\latinencoding{OT1}%
707         \ifx\cf@encoding\bbl@t@one
708         \xdef\latinencoding{\bbl@t@one}%
709         \else
710         \ifx\@fontenc@load@list\@undefined
711         \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}{}}%
712         \else
713         \def\@elt#1{, #1,}%
714         \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
715         \let\@elt\relax
716         \bbl@xin@{, T1, }\bbl@tempa
717         \ifin@
718         \xdef\latinencoding{\bbl@t@one}%
719         \fi
720         \fi
721     \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.

```

722 \DeclareRobustCommand{\latintext}{%
723     \fontencoding{\latinencoding}\selectfont
724     \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.

```

725 \ifx\@undefined\DeclareTextFontCommand
726     \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}

```

```

727 \else
728   \DeclareTextFontCommand{\textlatin}{\latintext}
729 \fi

```

## 7.9 Basic bidi support

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

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

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

- `pdftex` provides a minimal support for bidi text, and it must be done by hand. Vertical typesetting is not possible.
- `xetex` is somewhat better, thanks to its font engine (even if not always reliable) and a few additional tools. However, very little is done at the paragraph level. Another challenging problem is text direction does not honour  $\TeX$  grouping.
- `luatex` can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but bidi text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As `Lua $\TeX$ -ja` shows, vertical typesetting is possible, too.

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  $\LaTeX$ . Just in case, consider the possibility it has not been loaded.

```

730 \ifodd\bbl@engine
731   \def\bbl@activate@preotf{%
732     \let\bbl@activate@preotf\relax % only once
733     \directlua{
734       Babel = Babel or {}
735       %
736       function Babel.pre_otfload_v(head)
737         if Babel.numbers and Babel.digits_mapped then
738           head = Babel.numbers(head)
739         end
740         if Babel.bidi_enabled then
741           head = Babel.bidi(head, false, dir)
742         end
743         return head
744       end
745       %
746       function Babel.pre_otfload_h(head, gc, sz, pt, dir)
747         if Babel.numbers and Babel.digits_mapped then
748           head = Babel.numbers(head)
749         end
750         if Babel.bidi_enabled then
751           head = Babel.bidi(head, false, dir)
752         end
753         return head
754       end
755       %
756       luatexbase.add_to_callback('pre_linebreak_filter',
757         Babel.pre_otfload_v,
758         'Babel.pre_otfload_v',

```

```

759     luatexbase.priority_in_callback('pre_linebreak_filter',
760     'luaotfload.node_processor') or nil)
761 %
762     luatexbase.add_to_callback('hpack_filter',
763     Babel.pre_otfload_h,
764     'Babel.pre_otfload_h',
765     luatexbase.priority_in_callback('hpack_filter',
766     'luaotfload.node_processor') or nil)
767 }}
768 \fi

```

The basic setup. In luatex, the output is modified at a very low level to set the \bodydir to the \pagedir.

```

769 \bbl@trace{Loading basic (internal) bidi support}
770 \ifodd\bbl@engine
771   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
772     \let\bbl@beforeforeign\leavevmode
773     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
774     \RequirePackage{luatexbase}
775     \bbl@activate@preotf
776     \directlua{
777       require('babel-data-bidi.lua')
778       \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
779       require('babel-bidi-basic.lua')
780       \or
781       require('babel-bidi-basic-r.lua')
782     \fi}
783 % TODO - to locale_props, not as separate attribute
784 \newattribute\bbl@attr@dir
785 % TODO. I don't like it, hackish:
786 \bbl@exp{\output{\bodydir\pagedir\the\output}}
787 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
788 \fi\fi
789 \else
790   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
791     \bbl@error
792     {The bidi method 'basic' is available only in\\%
793     luatex. I'll continue with 'bidi=default', so\\%
794     expect wrong results}%
795     {See the manual for further details.}%
796     \let\bbl@beforeforeign\leavevmode
797     \AtEndOfPackage{%
798       \EnableBabelHook{babel-bidi}%
799       \bbl@xebidipar}
800   \fi\fi
801   \def\bbl@loadxebidi#1{%
802     \ifx\RTLfootnotetext\@undefined
803       \AtEndOfPackage{%
804         \EnableBabelHook{babel-bidi}%
805         \ifx\fontspec\@undefined
806           \bbl@loadfontspec % bidi needs fontspec
807         \fi
808         \usepackage#1{bidi}}%
809     \fi}
810   \ifnum\bbl@bidimode>200
811     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
812     \bbl@tentative{bidi=bidi}
813     \bbl@loadxebidi{}
814   \or

```

```

815 \bbl@loadxebidi{[rldocument]}
816 \or
817 \bbl@loadxebidi{}
818 \fi
819 \fi
820 \fi
821 \ifnum\bbl@bidimode=\@ne
822 \let\bbl@beforeforeign\leavevmode
823 \ifodd\bbl@engine
824 \newattribute\bbl@attr@dir
825 \bbl@exp{\output{\bodydir\pagedir\the\output}}%
826 \fi
827 \AtEndOfPackage{%
828 \EnableBabelHook{babel-bidi}%
829 \ifodd\bbl@engine\else
830 \bbl@xebidipar
831 \fi}
832 \fi

```

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

```

833 \bbl@trace{Macros to switch the text direction}
834 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
835 \def\bbl@rscripts{% TODO. Base on codes ??
836 ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
837 Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
838 Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
839 Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
840 Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
841 Old South Arabian,}%
842 \def\bbl@provide@dirs#1{%
843 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
844 \ifin@
845 \global\bbl@csarg\chardef{wdir@#1}\@ne
846 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
847 \ifin@
848 \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
849 \fi
850 \else
851 \global\bbl@csarg\chardef{wdir@#1}\z@
852 \fi
853 \ifodd\bbl@engine
854 \bbl@csarg\ifcase{wdir@#1}%
855 \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
856 \or
857 \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
858 \or
859 \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
860 \fi
861 \fi}
862 \def\bbl@switchdir{%
863 \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
864 \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
865 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}%
866 \def\bbl@setdirs#1{% TODO - math
867 \ifcase\bbl@select@type % TODO - strictly, not the right test
868 \bbl@bodydir{#1}%
869 \bbl@paddir{#1}%
870 \fi

```

```

871 \bbl@texdir{#1}}
872 % TODO. Only if \bbl@bidimode > 0?:
873 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
874 \DisableBabelHook{babel-bidi}

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

875 \ifodd\bbl@engine % luatex=1
876 \chardef\bbl@thetextdir\z@
877 \chardef\bbl@thepardir\z@
878 \def\bbl@getluadir#1{%
879   \directlua{
880     if tex.#1dir == 'TLT' then
881       tex.sprint('0')
882     elseif tex.#1dir == 'TRT' then
883       tex.sprint('1')
884     end}}
885 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\texdir.. 3=0 lr/1 rl
886   \ifcase#3\relax
887     \ifcase\bbl@getluadir{#1}\relax\else
888       #2 TLT\relax
889     \fi
890   \else
891     \ifcase\bbl@getluadir{#1}\relax
892       #2 TRT\relax
893     \fi
894   \fi}
895 \def\bbl@texdir#1{%
896   \bbl@setluadir{tex}\texdir{#1}%
897   \chardef\bbl@thetextdir#1\relax
898   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
899 \def\bbl@pardir#1{%
900   \bbl@setluadir{par}\pardir{#1}%
901   \chardef\bbl@thepardir#1\relax}
902 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
903 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
904 \def\bbl@dirparastext{\pardir\the\texdir\relax}% %%%
905 % Sadly, we have to deal with boxes in math with basic.
906 % Activated every math with the package option bidi=:
907 \ifnum\bbl@bidimode>\z@
908   \def\bbl@mathboxdir{%
909     \ifcase\bbl@thetextdir\relax
910       \everyhbox{\bbl@mathboxdir@aux L}%
911     \else
912       \everyhbox{\bbl@mathboxdir@aux R}%
913     \fi}
914   \def\bbl@mathboxdir@aux#1{%
915     \@ifnextchar\egroup{}\texdir T#1T\relax}}
916   \frozen@everymath\expandafter{%
917     \expandafter\bbl@mathboxdir\the\frozen@everymath}
918   \frozen@everydisplay\expandafter{%
919     \expandafter\bbl@mathboxdir\the\frozen@everydisplay}
920   \fi
921 \else % pdftex=0, xetex=2
922   \newcount\bbl@dirlevel
923   \chardef\bbl@thetextdir\z@
924   \chardef\bbl@thepardir\z@
925   \def\bbl@texdir#1{%
926     \ifcase#1\relax
927       \chardef\bbl@thetextdir\z@

```

```

928     \bbl@textdir@i\beginL\endL
929   \else
930     \chardef\bbl@thetextdir\@ne
931     \bbl@textdir@i\beginR\endR
932   \fi}
933 \def\bbl@textdir@i#1#2{%
934   \ifhmode
935     \ifnum\currentgrouplevel>\z@
936       \ifnum\currentgrouplevel=\bbl@dirlevel
937         \bbl@error{Multiple bidi settings inside a group}%
938         {I'll insert a new group, but expect wrong results.}%
939         \bgroup\aftergroup#2\aftergroup\egroup
940       \else
941         \ifcase\currentgrouptype\or % 0 bottom
942           \aftergroup#2% 1 simple {}
943         \or
944           \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
945         \or
946           \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
947         \or\or\or % vbox vtop align
948         \or
949           \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
950         \or\or\or\or\or\or % output math disc insert vcent mathchoice
951         \or
952           \aftergroup#2% 14 \begingroup
953         \else
954           \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
955         \fi
956       \fi
957       \bbl@dirlevel\currentgrouplevel
958     \fi
959   #1%
960   \fi}
961 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
962 \let\bbl@bodydir\@gobble
963 \let\bbl@pagedir\@gobble
964 \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).

```

965 \def\bbl@xebidipar{%
966   \let\bbl@xebidipar\relax
967   \TeXeTstate\@ne
968   \def\bbl@xeeverypar{%
969     \ifcase\bbl@thepardir
970       \ifcase\bbl@thetextdir\else\beginR\fi
971     \else
972       {\setbox\z@\lastbox\beginR\box\z@}%
973     \fi}%
974   \let\bbl@severypar\everypar
975   \newtoks\everypar
976   \everypar=\bbl@severypar
977   \bbl@severypar{\bbl@xeeverypar\the\everypar}}
978 \ifnum\bbl@bidimode>200
979   \let\bbl@textdir@i\@gobbletwo
980   \let\bbl@xebidipar\@empty
981   \AddBabelHook{bidi}{foreign}{%
982     \def\bbl@tempa{\def\BabelText####1}%

```

```

983 \ifcase\bbl@thetextdir
984 \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
985 \else
986 \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
987 \fi}
988 \def\bbl@paddir#1{\ifcase#1\relax\setLR\else\setRL\fi}
989 \fi
990 \fi

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

991 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
992 \AtBeginDocument{%
993 \ifx\pdfstringdefDisableCommands\@undefined\else
994 \ifx\pdfstringdefDisableCommands\relax\else
995 \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
996 \fi
997 \fi}

```

## 7.10 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`.

```

998 \bbl@trace{Local Language Configuration}
999 \ifx\loadlocalcfg\@undefined
1000 \@ifpackagewith{babel}{noconfigs}%
1001 {\let\loadlocalcfg\@gobble}%
1002 {\def\loadlocalcfg#1{%
1003 \InputIfFileExists{#1.cfg}%
1004 {\typeout{*****^J%
1005 * Local config file #1.cfg used^^J%
1006 *}}}%
1007 \@empty}}
1008 \fi

```

## 7.11 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).

```

1009 \bbl@trace{Language options}
1010 \let\bbl@afterlang\relax
1011 \let\BabelModifiers\relax
1012 \let\bbl@loaded\@empty
1013 \def\bbl@load@language#1{%
1014 \InputIfFileExists{#1.ldf}%
1015 {\edef\bbl@loaded{\CurrentOption
1016 \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
1017 \expandafter\let\expandafter\bbl@afterlang
1018 \csname\CurrentOption.ldf-h@k\endcsname
1019 \expandafter\let\expandafter\BabelModifiers
1020 \csname bbl@mod@\CurrentOption\endcsname}%
1021 {\bbl@error{%
1022 Unknown option '\CurrentOption'. Either you misspelled it\\%
1023 or the language definition file \CurrentOption.ldf was not found}}%
1024 Valid options are, among others: shorthands=, KeepShorthandsActive,\\%

```

```

1025     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
1026     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.

```

1027 \def\bbl@try@load@lang#1#2#3{%
1028   \IfFileExists{\CurrentOption.lda}%
1029   {\bbl@load@language{\CurrentOption}}}%
1030   {#1\bbl@load@language{#2}#3}}
1031 \DeclareOption{hebrew}{%
1032   \input{rlbabel.def}%
1033   \bbl@load@language{hebrew}}
1034 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
1035 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
1036 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
1037 \DeclareOption{polutonikogreek}{%
1038   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}%
1039 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
1040 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
1041 \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 .lda 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.

```

1042 \ifx\bbl@opt@config\@nnil
1043   \@ifpackagewith{babel}{noconfigs}{}%
1044   {\InputIfFileExists{bblopts.cfg}%
1045     {\typeout{*****^J%
1046               * Local config file bblopts.cfg used^^J%
1047               *}}}%
1048   {}}%
1049 \else
1050   \InputIfFileExists{\bbl@opt@config.cfg}%
1051   {\typeout{*****^J%
1052             * Local config file \bbl@opt@config.cfg used^^J%
1053             *}}}%
1054   {\bbl@error{%
1055     Local config file '\bbl@opt@config.cfg' not found}{%
1056     Perhaps you misspelled it.}}}%
1057 \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 (note this list also contains the language given with main). If not declared above, the names of the option and the file are the same.

```

1058 \let\bbl@tempc\relax
1059 \bbl@foreach\bbl@language@opts{%
1060   \ifcase\bbl@iniflag % Default
1061     \bbl@ifunset{ds@#1}%
1062     {\DeclareOption{#1}{\bbl@load@language{#1}}}%
1063     {}%
1064   \or % provide=*
1065     \@gobble % case 2 same as 1
1066   \or % provide+=*
1067     \bbl@ifunset{ds@#1}%
1068     {\IfFileExists{#1.lda}{%
1069       {\IfFileExists{babel-#1.tex}{\@namedef{ds@#1}}}}}%

```



```

1070     {}%
1071     \bbl@ifunset{ds@#1}%
1072     {\def\bbl@tempc{#1}%
1073      \DeclareOption{#1}{%
1074        \ifnum\bbl@iniflag>\@ne
1075          \bbl@ldfinit
1076          \babelprovide[import]{#1}%
1077          \bbl@afterldf{}%
1078        \else
1079          \bbl@load@language{#1}%
1080        \fi}}%
1081     {}%
1082 \or    % provide*=*
1083 \def\bbl@tempc{#1}%
1084 \bbl@ifunset{ds@#1}%
1085     {\DeclareOption{#1}{%
1086       \bbl@ldfinit
1087       \babelprovide[import]{#1}%
1088       \bbl@afterldf{}}}%
1089     {}%
1090 \fi}

```

Now, we make sure an option is explicitly declared for any language set as global option, by checking if an ldf exists. The previous step was, in fact, somewhat redundant, but that way we minimize accessing the file system just to see if the option could be a language.

```

1091 \let\bbl@tempb\@nnil
1092 \bbl@foreach\@classoptionslist{%
1093   \bbl@ifunset{ds@#1}%
1094   {\IfFileExists{#1.ldf}%
1095    {\def\bbl@tempb{#1}%
1096     \DeclareOption{#1}{%
1097       \ifnum\bbl@iniflag>\@ne
1098         \bbl@ldfinit
1099         \babelprovide[import]{#1}%
1100         \bbl@afterldf{}%
1101       \else
1102         \bbl@load@language{#1}%
1103       \fi}}%
1104   {\IfFileExists{babel-#1.tex}% TODO. Copypaste pattern
1105    {\def\bbl@tempb{#1}%
1106     \DeclareOption{#1}{%
1107       \ifnum\bbl@iniflag>\@ne
1108         \bbl@ldfinit
1109         \babelprovide[import]{#1}%
1110         \bbl@afterldf{}%
1111       \else
1112         \bbl@load@language{#1}%
1113       \fi}}%
1114    {}}}%
1115   {}}

```

If a main language has been set, store it for the third pass.

```

1116 \ifnum\bbl@iniflag=\z@\else
1117   \ifx\bbl@opt@main\@nnil
1118     \ifx\bbl@tempc\relax
1119       \let\bbl@opt@main\bbl@tempb
1120     \else
1121       \let\bbl@opt@main\bbl@tempc
1122     \fi

```

```

1123 \fi
1124 \fi
1125 \ifx\bbl@opt@main\@nnil\else
1126 \expandafter
1127 \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
1128 \expandafter\let\csname ds@\bbl@opt@main\endcsname\empty
1129 \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 (except, of course, global options, which  $\TeX$  processes before):

```

1130 \def\AfterBabelLanguage#1{%
1131 \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
1132 \DeclareOption*{}
1133 \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. Then execute directly the option (because it could be used only in main). After loading all languages, we deactivate \AfterBabelLanguage.

```

1134 \bbl@trace{Option 'main'}
1135 \ifx\bbl@opt@main\@nnil
1136 \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
1137 \let\bbl@tempc\empty
1138 \bbl@for\bbl@tempb\bbl@tempa{%
1139 \bbl@xin@{\bbl@tempb,}{,\bbl@loaded,}%
1140 \ifin@{\edef\bbl@tempc{\bbl@tempb}}\fi}
1141 \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
1142 \expandafter\bbl@tempa\bbl@loaded,\@nnil
1143 \ifx\bbl@tempb\bbl@tempc\else
1144 \bbl@warning{%
1145 Last declared language option is '\bbl@tempc',\%
1146 but the last processed one was '\bbl@tempb'.\%
1147 The main language can't be set as both a global\%
1148 and a package option. Use 'main=\bbl@tempc' as\%
1149 option. Reported}%
1150 \fi
1151 \else
1152 \ifodd\bbl@iniflag % case 1,3
1153 \bbl@ldfinit
1154 \let\CurrentOption\bbl@opt@main
1155 \ifx\bbl@opt@provide\@nnil
1156 \bbl@exp{\bbl@babelprovide[import,main]{\bbl@opt@main}}
1157 \else
1158 \bbl@exp{\bbl@babelprovide[\bbl@opt@provide,main]{\bbl@opt@main}}%
1159 \fi
1160 \bbl@afterldf{}%
1161 \else % case 0,2
1162 \chardef\bbl@iniflag\z@ % Force ldf
1163 \expandafter\let\csname ds@\bbl@opt@main\endcsname\bbl@loadmain
1164 \ExecuteOptions{\bbl@opt@main}
1165 \DeclareOption*{}%
1166 \ProcessOptions*
1167 \fi
1168 \fi
1169 \def\AfterBabelLanguage{%
1170 \bbl@error
1171 {Too late for \string\AfterBabelLanguage}%

```

```

1172   {Languages have been loaded, so I can do nothing}}

In order to catch the case where the user forgot to specify a language we check whether
\bb1@main@language, has become defined. If not, no language has been loaded and an error
message is displayed.

1173 \ifx\bb1@main@language\@undefined
1174   \bb1@info{%
1175     You haven't specified a language. I'll use 'nil'\%
1176     as the main language. Reported}
1177   \bb1@load@language{nil}
1178 \fi
1179 </package>
1180 <*core>

```

## 8 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 T<sub>E</sub>X users might want to use some of the features of the babel system too, care has to be taken that plain T<sub>E</sub>X 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 T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X, some of it is for the L<sup>A</sup>T<sub>E</sub>X 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.

### 8.1 Tools

```

1181 \ifx\ldf@quit\@undefined\else
1182 \endinput\fi % Same line!
1183 <<Make sure ProvidesFile is defined>>
1184 \ProvidesFile{babel.def}[(<date>)] <<version>> Babel common definitions]

```

The file babel.def expects some definitions made in the L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> style file. So, In L<sup>A</sup>T<sub>E</sub>X 2.09 and 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).

```

1185 \ifx\AtBeginDocument\@undefined % TODO. change test.
1186   <<Emulate LaTeX>>
1187   \def\languagename{english}%
1188   \let\bb1@opt@shorthands\@nnil
1189   \def\bb1@ifshorthand#1#2#3{#2}%
1190   \let\bb1@language@opts\@empty
1191   \ifx\babeloptionstrings\@undefined
1192     \let\bb1@opt@strings\@nnil
1193   \else
1194     \let\bb1@opt@strings\babeloptionstrings
1195   \fi
1196   \def\BabelStringsDefault{generic}
1197   \def\bb1@tempa{normal}
1198   \ifx\babeloptionmath\bb1@tempa
1199     \def\bb1@mathnormal{\noexpand\textormath}
1200   \fi
1201   \def\AfterBabelLanguage#1#2{}
1202   \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
1203   \let\bb1@afterlang\relax
1204   \def\bb1@opt@safe{BR}

```

```

1205 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
1206 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
1207 \expandafter\newif\csname ifbbl@single\endcsname
1208 \chardef\bbl@bidimode\z@
1209 \fi

```

Exit immediately with 2.09. An error is raised by the sty file, but also try to minimize the number of errors.

```

1210 \ifx\bbl@trace\@undefined
1211 \let\LdfInit\endinput
1212 \def\ProvidesLanguage#1\endinput}
1213 \endinput\fi % Same line!

```

And continue.

## 9 Multiple languages

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

Plain T<sub>E</sub>X 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.

```

1214 <<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.

```

1215 \def\bbl@version{<<version>>}
1216 \def\bbl@date{<<date>>}
1217 \def\adddialect#1#2{%
1218   \global\chardef#1#2\relax
1219   \bbl@usehooks{adddialect}{#1}{#2}}%
1220 \begingroup
1221   \count#1\relax
1222   \def\bbl@elt##1##2###3###4{%
1223     \ifnum\count@=##2\relax
1224       \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
1225       \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
1226               set to \expandafter\string\csname l@##1\endcsname\\%
1227               (\string\language\the\count@). Reported}%
1228       \def\bbl@elt####1####2####3####4{}}%
1229   \fi}%
1230   \bbl@cs{languages}%
1231 \endgroup

```

`\bbl@iflanguage` executes code only if the language `l@` exists. Otherwise raises and 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 intended to fix a long-standing bug when `\foreignlanguage` and the like appear in a `\MakeXXcase`. 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.

```

1232 \def\bbl@fixname#1{%
1233   \begingroup
1234   \def\bbl@tempe{l@}%
1235   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
1236   \bbl@tempd
1237   {\lowercase\expandafter\bbl@tempd}%
1238   {\uppercase\expandafter\bbl@tempd}%
1239   \@empty
1240   {\edef\bbl@tempd{\def\noexpand#1{#1}}}%
1241   {\uppercase\expandafter\bbl@tempd}}}%
1242   {\edef\bbl@tempd{\def\noexpand#1{#1}}}%
1243   {\lowercase\expandafter\bbl@tempd}}}%

```

```

1244 \empty
1245 \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
1246 \bbl@tempd
1247 \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}%
1248 \def\bbl@iflanguage#1{%
1249 \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.

```

1250 \def\bbl@bcpcase#1#2#3#4\@#5{%
1251 \ifx\empty#3%
1252 \uppercase{\def#5{#1#2}}%
1253 \else
1254 \uppercase{\def#5{#1}}%
1255 \lowercase{\edef#5{#5#2#3#4}}%
1256 \fi}
1257 \def\bbl@bcpllookup#1-#2-#3-#4\@{%
1258 \let\bbl@bcp\relax
1259 \lowercase{\def\bbl@tempa{#1}}%
1260 \ifx\empty#2%
1261 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1262 \else\ifx\empty#3%
1263 \bbl@bcpcase#2\empty\empty\@bbl@tempb
1264 \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
1265 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
1266 {}%
1267 \ifx\bbl@bcp\relax
1268 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1269 \fi
1270 \else
1271 \bbl@bcpcase#2\empty\empty\@bbl@tempb
1272 \bbl@bcpcase#3\empty\empty\@bbl@tempc
1273 \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
1274 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
1275 {}%
1276 \ifx\bbl@bcp\relax
1277 \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1278 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1279 {}%
1280 \fi
1281 \ifx\bbl@bcp\relax
1282 \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1283 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1284 {}%
1285 \fi
1286 \ifx\bbl@bcp\relax
1287 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1288 \fi
1289 \fi\fi}
1290 \let\bbl@initoload\relax
1291 \def\bbl@provide@locale{%
1292 \ifx\babelprovide\undefined
1293 \bbl@error{For a language to be defined on the fly 'base'\%
1294 is not enough, and the whole package must be\%
1295 loaded. Either delete the 'base' option or\%
1296 request the languages explicitly}%

```

```

1297             {See the manual for further details.}%
1298 \fi
1299 % TODO. Option to search if loaded, with \LocaleForEach
1300 \let\bbbl@auxname\language % Still necessary. TODO
1301 \bbbl@ifunset{\bbbl@bcp@map@\language}{}% Move uplevel??
1302 { \edef\language{\@nameuse{\bbbl@bcp@map@\language}}}%
1303 \ifbbbl@bcp@allowed
1304 \expandafter\ifx\csname date\language\endcsname\relax
1305 \expandafter
1306 \bbbl@bcp@lookup\language-\@empty-\@empty-\@empty\@
1307 \ifx\bbbl@bcp\relax\else % Returned by \bbbl@bcp@lookup
1308 \edef\language{\bbbl@bcp@prefix\bbbl@bcp}%
1309 \edef\localename{\bbbl@bcp@prefix\bbbl@bcp}%
1310 \expandafter\ifx\csname date\language\endcsname\relax
1311 \let\bbbl@initoload\bbbl@bcp
1312 \bbbl@exp{\babelprovide[\bbbl@autoload@bcptoptions]{\language}}%
1313 \let\bbbl@initoload\relax
1314 \fi
1315 \bbbl@csarg\xdef{bcp@map@\bbbl@bcp}{\localename}%
1316 \fi
1317 \fi
1318 \fi
1319 \expandafter\ifx\csname date\language\endcsname\relax
1320 \IfFileExists{babel-\language.tex}%
1321 {\bbbl@exp{\babelprovide[\bbbl@autoload@options]{\language}}}%
1322 {}%
1323 \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.

```

1324 \def\iflanguage#1{%
1325 \bbbl@iflanguage{#1}{%
1326 \ifnum\csname l@#1\endcsname=\language
1327 \expandafter\@firstoftwo
1328 \else
1329 \expandafter\@secondoftwo
1330 \fi}}

```

## 9.1 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.

```

1331 \let\bbbl@select@type\z@
1332 \edef\selectlanguage{%
1333 \noexpand\protect
1334 \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`.

```
1335 \ifx\@undefined\protect\let\protect\relax\fi
```

The following definition is preserved for backwards compatibility. It is related to a trick for 2.09.

```
1336 \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.

```
1337 \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`

```
1338 \def\bbl@push@language{%
1339   \ifx\language\@undefined\else
1340     \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1341   \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`.

```
1342 \def\bbl@pop@lang#1+#2\@@{%
1343   \edef\language{#1}%
1344   \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).

```
1345 \let\bbl@ifrestoring\@secondoftwo
1346 \def\bbl@pop@language{%
1347   \expandafter\bbl@pop@lang\bbl@language@stack\@@
1348   \let\bbl@ifrestoring\@firstoftwo
1349   \expandafter\bbl@set@language\expandafter{\language}%
1350   \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).

```
1351 \chardef\localeid\z@
1352 \def\bbl@id@last{0} % No real need for a new counter
1353 \def\bbl@id@assign{%
1354   \bbl@ifunset{bbl@id@@\language}%
1355   {\count@ \bbl@id@last \relax
1356    \advance\count@ \one
1357    \bbl@csarg\chardef{id@@\language}\count@
1358    \edef\bbl@id@last{\the\count@}%
1359    \ifcase\bbl@engine\or
1360      \directlua{
1361        Babel = Babel or {}
```

```

1362      Babel.locale_props = Babel.locale_props or {}
1363      Babel.locale_props[\bbl@id@last] = {}
1364      Babel.locale_props[\bbl@id@last].name = '\language'
1365    }%
1366    \fi}%
1367  }%
1368  \chardef\localeid\bbl@c1{id@}}

```

The unprotected part of \selectlanguage.

```

1369 \expandafter\def\csname selectlanguage \endcsname#1{%
1370   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@\fi
1371   \bbl@push@language
1372   \aftergroup\bbl@pop@language
1373   \bbl@set@language{#1}}

```

\bbl@set@language The macro \bbl@set@language takes care of switching the language environment *and* of writing entries on the auxiliary files. For historical reasons, language names can be either language of \language. To catch either form a trick is used, but unfortunately as a side effect the catcodes of letters in \language are messed up. This is a bug, but preserved for backwards compatibility. The list of auxiliary files can be extended by redefining \BabelContentsFiles, but make sure they are loaded inside a group (as aux, toc, lof, and lot do) or the last language of the document will remain active afterwards.

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

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

```

1374 \def\BabelContentsFiles{toc,lof,lot}
1375 \def\bbl@set@language#1{% from selectlanguage, pop@
1376   % The old buggy way. Preserved for compatibility.
1377   \edef\language{%
1378     \ifnum\escapechar=\expandafter`\string#1\@empty
1379     \else\string#1\@empty\fi}%
1380   \ifcat\relax\noexpand#1%
1381     \expandafter\ifx\csname date\language\endcsname\relax
1382       \edef\language{#1}%
1383       \let\localename\language
1384     \else
1385       \bbl@info{Using '\string\language' instead of 'language' is\\%
1386         deprecated. If what you want is to use a\\%
1387         macro containing the actual locale, make\\%
1388         sure it does not not match any language.\\%
1389         Reported}%
1390       \ifx\scantokens\undefined
1391         \def\localename{??}%
1392       \else
1393         \scantokens\expandafter{\expandafter
1394           \def\expandafter\localename\expandafter{\language}}%
1395       \fi
1396     \fi
1397   \else
1398     \def\localename{#1}% This one has the correct catcodes
1399   \fi
1400   \select@language{\language}%
1401   % write to auxs
1402   \expandafter\ifx\csname date\language\endcsname\relax\else
1403     \if@filesw
1404       \ifx\babel@aux@gobbletwo\else % Set if single in the first, redundant
1405         \bbl@savelastskip

```



```

1406      \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
1407      \bbl@restorelastskip
1408      \fi
1409      \bbl@usehooks{write}{}%
1410      \fi
1411    \fi}
1412 %
1413 \let\bbl@restorelastskip\relax
1414 \def\bbl@savelastskip{%
1415   \let\bbl@restorelastskip\relax
1416   \ifvmode
1417     \ifdim\lastskip=\z@
1418       \let\bbl@restorelastskip\nobreak
1419     \else
1420       \bbl@exp{%
1421         \def\\bbl@restorelastskip{%
1422           \skip@=\the\lastskip
1423           \\nobreak \vskip-\skip@ \vskip\skip@}}%
1424       \fi
1425     \fi}
1426 %
1427 \newif\ifbbl@bcpallowed
1428 \bbl@bcpallowedfalse
1429 \def\select@language#1{% from set@, babel@aux
1430   % set hmap
1431   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1432   % set name
1433   \edef\language#1}%
1434   \bbl@fixname\language
1435   % TODO. name@map must be here?
1436   \bbl@provide@locale
1437   \bbl@iflanguage\language
1438     \expandafter\ifx\csgname date\language\endcsname\relax
1439     \bbl@error
1440       {Unknown language '\language'. Either you have\\%
1441       misspelled its name, it has not been installed,\\%
1442       or you requested it in a previous run. Fix its name,\\%
1443       install it or just rerun the file, respectively. In\\%
1444       some cases, you may need to remove the aux file}%
1445       {You may proceed, but expect wrong results}%
1446   \else
1447     % set type
1448     \let\bbl@select@type\z@
1449     \expandafter\bbl@switch\expandafter{\language}%
1450     \fi}}
1451 \def\babel@aux#1#2{% TODO. See how to avoid undefined nil's
1452   \select@language{#1}%
1453   \bbl@foreach\BabelContentsFiles{%
1454     \@writefile{##1}{\babel@toc{#1}{#2}}}% % TODO - ok in plain?
1455 \def\babel@toc#1#2{%
1456   \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 `\csgname` primitive. Now activate the language-specific definitions. This is done by constructing the names of three

macros by concatenating three words with the argument of `\selectlanguage`, and calling these macros.

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

```

1457 \newif\ifbbl@usedategroup
1458 \def\bbl@switch#1{% from select@, foreign@
1459 % make sure there is info for the language if so requested
1460 \bbl@ensureinfo{#1}%
1461 % restore
1462 \originalTeX
1463 \expandafter\def\expandafter\originalTeX\expandafter{%
1464 \csname noextras#1\endcsname
1465 \let\originalTeX\@empty
1466 \babel@beginsave}%
1467 \bbl@usehooks{afterreset}{}%
1468 \languageshorthands{none}%
1469 % set the locale id
1470 \bbl@id@assign
1471 % switch captions, date
1472 % No text is supposed to be added here, so we remove any
1473 % spurious spaces.
1474 \bbl@bsphack
1475 \ifcase\bbl@select@type
1476 \csname captions#1\endcsname\relax
1477 \csname date#1\endcsname\relax
1478 \else
1479 \bbl@xin@{,captions,}{, \bbl@select@opts,}%
1480 \ifin@
1481 \csname captions#1\endcsname\relax
1482 \fi
1483 \bbl@xin@{,date,}{, \bbl@select@opts,}%
1484 \ifin@ % if \foreign... within \langle lang \rangle date
1485 \csname date#1\endcsname\relax
1486 \fi
1487 \fi
1488 \bbl@esphack
1489 % switch extras
1490 \bbl@usehooks{beforeextras}{}%
1491 \csname extras#1\endcsname\relax
1492 \bbl@usehooks{afterextras}{}%
1493 % > babel-ensure
1494 % > babel-sh-<short>
1495 % > babel-bidi
1496 % > babel-fontspec
1497 % hyphenation - case mapping
1498 \ifcase\bbl@opt@hyphenmap\or
1499 \def\BabelLower##1##2{\lccode##1=##2\relax}%
1500 \ifnum\bbl@hymapsel>4\else
1501 \csname\language @bbl@hyphenmap\endcsname
1502 \fi
1503 \chardef\bbl@opt@hyphenmap\z@
1504 \else
1505 \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
1506 \csname\language @bbl@hyphenmap\endcsname
1507 \fi
1508 \fi
1509 \let\bbl@hymapsel\@cclv

```

```

1510 % hyphenation - select rules
1511 \ifnum\csname l@\language\endcsname=\l@unhyphenated
1512   \edef\bbl@tempa{u}%
1513   \else
1514     \edef\bbl@tempa{\bbl@cl{lnbrk}}%
1515   \fi
1516 % linebreaking - handle u, e, k (v in the future)
1517 \bbl@xin@{/u}{/\bbl@tempa}%
1518 \ifin@else\bbl@xin@{/e}{/\bbl@tempa}\fi % elongated forms
1519 \ifin@else\bbl@xin@{/k}{/\bbl@tempa}\fi % only kashida
1520 \ifin@else\bbl@xin@{/v}{/\bbl@tempa}\fi % variable font
1521 \ifin@
1522   % unhyphenated/kashida/elongated = allow stretching
1523   \language\l@unhyphenated
1524   \babel@savevariable\emergencystretch
1525   \emergencystretch\maxdimen
1526   \babel@savevariable\hbadness
1527   \hbadness\@M
1528 \else
1529   % other = select patterns
1530   \bbl@patterns{#1}%
1531 \fi
1532 % hyphenation - mins
1533 \babel@savevariable\lefthyphenmin
1534 \babel@savevariable\righthyphenmin
1535 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1536   \set@hyphenmins\tw@\thr@@\relax
1537 \else
1538   \expandafter\expandafter\expandafter\set@hyphenmins
1539     \csname #1hyphenmins\endcsname\relax
1540 \fi}

```

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

The \ignorespaces command is necessary to hide the environment when it is entered in horizontal mode.

```

1541 \long\def\otherlanguage#1{%
1542   \ifnum\bbl@hymapsel=\@cc\l\let\bbl@hymapsel\thr@@\fi
1543   \csname selectlanguage \endcsname{#1}%
1544   \ignorespaces}

```

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

```

1545 \long\def\endotherlanguage{%
1546   \global\@ignoretrue\ignorespaces}

```

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

```

1547 \expandafter\def\csname otherlanguage*\endcsname{%
1548   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}%
1549   \def\bbl@otherlanguage@s[#1]#2{%
1550     \ifnum\bbl@hymapsel=\@cc\l\chardef\bbl@hymapsel4\relax\fi
1551     \def\bbl@select@opts{#1}%
1552     \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”.

```
1553 \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.

```
1554 \providecommand\bbl@beforeforeign{}
1555 \edef\foreignlanguage{%
1556   \noexpand\protect
1557   \expandafter\noexpand\csname foreignlanguage \endcsname}
1558 \expandafter\def\csname foreignlanguage \endcsname{%
1559   \@ifstar\bbl@foreign@s\bbl@foreign@x}
1560 \providecommand\bbl@foreign@x[3][]{%
1561   \begingroup
1562     \def\bbl@select@opts{#1}%
1563     \let\BabelText\@firstofone
1564     \bbl@beforeforeign
1565     \foreign@language{#2}%
1566     \bbl@usehooks{foreign}{}%
1567     \BabelText{#3}% Now in horizontal mode!
1568   \endgroup}
1569 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@par
1570   \begingroup
1571     {\par}%
1572     \let\bbl@select@opts\@empty
1573     \let\BabelText\@firstofone
1574     \foreign@language{#1}%
1575     \bbl@usehooks{foreign*}{}%
1576     \bbl@dirparastext
1577     \BabelText{#2}% Still in vertical mode!
1578     {\par}%
1579   \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`.

```
1580 \def\foreign@language#1{%
1581   % set name
1582   \edef\languagename{#1}%
```

```

1583 \ifbbl@usedategroup
1584   \bbl@add\bbl@select@opts{,date,}%
1585   \bbl@usedategroupfalse
1586 \fi
1587 \bbl@fixname\language
1588 % TODO. name@map here?
1589 \bbl@provide@locale
1590 \bbl@iflanguage\language{
1591   \expandafter\ifx\csname date\language\endcsname\relax
1592     \bbl@warning % TODO - why a warning, not an error?
1593     {Unknown language '#1'. Either you have\\%
1594      misspelled its name, it has not been installed,\\%
1595      or you requested it in a previous run. Fix its name,\\%
1596      install it or just rerun the file, respectively. In\\%
1597      some cases, you may need to remove the aux file.\\%
1598      I'll proceed, but expect wrong results.\\%
1599      Reported}%
1600 \fi
1601 % set type
1602 \let\bbl@select@type\@ne
1603 \expandafter\bbl@switch\expandafter{\language}}

```

**\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.

```

1604 \let\bbl@hyphlist\@empty
1605 \let\bbl@hyphenation@\relax
1606 \let\bbl@pttnlist\@empty
1607 \let\bbl@patterns@\relax
1608 \let\bbl@hymapsel=\@cclv
1609 \def\bbl@patterns#1{%
1610   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1611     \csname l@#1\endcsname
1612     \edef\bbl@tempa{#1}%
1613   \else
1614     \csname l@#1:\f@encoding\endcsname
1615     \edef\bbl@tempa{#1:\f@encoding}%
1616   \fi
1617   \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}}%
1618 % > luatex
1619 \@ifundefined{bbl@hyphenation@}{% Can be \relax!
1620   \begingroup
1621     \bbl@xin@{, \number\language,}{, \bbl@hyphlist}%
1622     \ifin@ \else
1623       \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}%
1624       \hyphenation{%
1625         \bbl@hyphenation@
1626         \@ifundefined{bbl@hyphenation@#1}%
1627         \@empty
1628         {\space\csname bbl@hyphenation@#1\endcsname}}%
1629       \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1630     \fi
1631   \endgroup}}

```

**hyphenrules** 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*`.

```

1632 \def\hyphenrules#1{%
1633   \edef\bbl@tempf{#1}%
1634   \bbl@fixname\bbl@tempf
1635   \bbl@iflanguage\bbl@tempf{%
1636     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1637     \ifx\languageshorthands\undefined\else
1638       \languageshorthands{none}%
1639     \fi
1640     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1641       \set@hyphenmins\tw@\thr@@\relax
1642     \else
1643       \expandafter\expandafter\expandafter\set@hyphenmins
1644       \csname\bbl@tempf hyphenmins\endcsname\relax
1645     \fi}}
1646 \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.

```

1647 \def\providehyphenmins#1#2{%
1648   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1649     \@namedef{#1hyphenmins}{#2}%
1650   \fi}

```

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

```

1651 \def\set@hyphenmins#1#2{%
1652   \lefthyphenmin#1\relax
1653   \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.

```

1654 \ifx\ProvidesFile\undefined
1655   \def\ProvidesLanguage#1[#2 #3 #4]{%
1656     \wlog{Language: #1 #4 #3 <#2>}%
1657   }
1658 \else
1659   \def\ProvidesLanguage#1{%
1660     \begingroup
1661     \catcode`\ 10 %
1662     \@makeother\/%
1663     \ifnextchar[%]
1664       {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
1665     \def\@provideslanguage#1[#2]{%
1666       \wlog{Language: #1 #2}%
1667       \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1668     }
1669   \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`.

```

1670 \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.

```
1671 \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’:

```
1672 \providecommand\setlocale{%
1673   \bbl@error
1674   {Not yet available}%
1675   {Find an armchair, sit down and wait}}
1676 \let\uselocale\setlocale
1677 \let\locale\setlocale
1678 \let\selectlocale\setlocale
1679 \let\localename\setlocale
1680 \let\textlocale\setlocale
1681 \let\textlanguage\setlocale
1682 \let\languagetext\setlocale
```

## 9.2 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.

```
1683 \edef\bbl@nulllanguage{\string\language=0}
1684 \ifx\PackageError\@undefined % TODO. Move to Plain
1685   \def\bbl@error#1#2{%
1686     \begingroup
1687       \newlinechar=`^^J
1688       \def\{^^J(babel) }%
1689       \errhelp{#2}\errmessage{\{#1}%
1690     \endgroup}
1691   \def\bbl@warning#1{%
1692     \begingroup
1693       \newlinechar=`^^J
1694       \def\{^^J(babel) }%
1695       \message{\{#1}%
1696     \endgroup}
1697   \let\bbl@infowarn\bbl@warning
1698   \def\bbl@info#1{%
1699     \begingroup
1700       \newlinechar=`^^J
1701       \def\{^^J}%
1702       \wlog{#1}%
1703     \endgroup}
1704 \fi
1705 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1706 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1707   \global\@namedef{#2}{\textbf{?#1?}}%
1708   \@nameuse{#2}%
1709   \edef\bbl@tempa{#1}%
1710   \bbl@sreplace\bbl@tempa{name}{}}%
```

```

1711 \bbl@warning{% TODO.
1712 \@backslashchar#1 not set for '\language'. Please,\\%
1713 define it after the language has been loaded\\%
1714 (typically in the preamble) with:\\%
1715 \string\setlocalecaption{\language}{\bbl@tempa}{..}\\%
1716 Reported}}
1717 \def\bbl@tentative{\protect\bbl@tentative@i}
1718 \def\bbl@tentative@i#1{%
1719 \bbl@warning{%
1720 Some functions for '#1' are tentative.\\%
1721 They might not work as expected and their behavior\\%
1722 could change in the future.\\%
1723 Reported}}
1724 \def\@nolanerr#1{%
1725 \bbl@error
1726 {You haven't defined the language '#1' yet.\\%
1727 Perhaps you misspelled it or your installation\\%
1728 is not complete}%
1729 {Your command will be ignored, type <return> to proceed}}
1730 \def\@nopatterns#1{%
1731 \bbl@warning
1732 {No hyphenation patterns were preloaded for\\%
1733 the language '#1' into the format.\\%
1734 Please, configure your TeX system to add them and\\%
1735 rebuild the format. Now I will use the patterns\\%
1736 preloaded for \bbl@nulllanguage\space instead}}
1737 \let\bbl@usehooks\@gobbletwo
1738 \ifx\bbl@onlyswitch\@empty\endinput\fi
1739 % Here ended switch.def

Here ended switch.def.

1740 \ifx\directlua\@undefined\else
1741 \ifx\bbl@luapatterns\@undefined
1742 \input luababel.def
1743 \fi
1744 \fi
1745 <<Basic macros>>
1746 \bbl@trace{Compatibility with language.def}
1747 \ifx\bbl@languages\@undefined
1748 \ifx\directlua\@undefined
1749 \openin1 = language.def % TODO. Remove hardcoded number
1750 \ifeof1
1751 \closein1
1752 \message{I couldn't find the file language.def}
1753 \else
1754 \closein1
1755 \begingroup
1756 \def\addlanguage#1#2#3#4#5{%
1757 \expandafter\ifx\csname lang@#1\endcsname\relax\else
1758 \global\expandafter\let\csname l@#1\endcsname
1759 \csname lang@#1\endcsname
1760 \fi}%
1761 \def\uselanguage#1{%
1762 \input language.def
1763 \endgroup
1764 \fi
1765 \fi
1766 \chardef\l@english\z@
1767 \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.

```
1768 \def\addto#1#2{%
1769   \ifx#1\@undefined
1770     \def#1{#2}%
1771   \else
1772     \ifx#1\relax
1773       \def#1{#2}%
1774     \else
1775       {\toks@\expandafter{#1#2}%
1776        \xdef#1{\the\toks@}}%
1777   \fi
1778 }
```

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. `TODO`. Always used with additional expansions. Move them here? Move the macro to basic?

```
1779 \def\bbl@withactive#1#2{%
1780   \begingroup
1781   \lccode`~=#2\relax
1782   \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`.

```
1783 \def\bbl@redefine#1{%
1784   \edef\bbl@tempa{\bbl@stripslash#1}%
1785   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1786   \expandafter\def\csname\bbl@tempa\endcsname{
1787 \@onlypreamble\bbl@redefine
```

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

```
1788 \def\bbl@redefine@long#1{%
1789   \edef\bbl@tempa{\bbl@stripslash#1}%
1790   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1791   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname{
1792 \@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_`.

```
1793 \def\bbl@redefineroobust#1{%
1794   \edef\bbl@tempa{\bbl@stripslash#1}%
1795   \bbl@ifunset{\bbl@tempa\space}%
1796   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1797    \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1798   {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1799   \@namedef{\bbl@tempa\space}%
1800 \@onlypreamble\bbl@redefineroobust
```

### 9.3 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.

```

1801 \bbl@trace{Hooks}
1802 \newcommand\AddBabelHook[3][]{%
1803   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1804   \def\bbl@tempa##1,##3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1805   \expandafter\bbl@tempa\bbl@evargs,##3=,\@empty
1806   \bbl@ifunset{bbl@ev@#2@#3@#1}%
1807     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1808     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1809   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1810 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1811 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1812 \def\bbl@usehooks#1#2{%
1813   \def\bbl@elth##1{%
1814     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@#2}}%
1815     \bbl@cs{ev@#1@}%
1816     \ifx\language\@undefined\else % Test required for Plain (?)
1817       \def\bbl@elth##1{%
1818         \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1@#2}}%
1819         \bbl@cl{ev@#1}%
1820       \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).

```

1821 \def\bbl@evargs{% <- don't delete this comma
1822   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1823   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1824   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1825   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1826   beforestart=0,language=2}

```

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

```

1827 \bbl@trace{Defining babelensure}
1828 \newcommand\babelensure[2][]{% TODO - revise test files
1829   \AddBabelHook{babel-ensure}{afterextras}{%
1830     \ifcase\bbl@select@type
1831       \bbl@cl{e}%
1832     \fi}%
1833   \begingroup
1834     \let\bbl@ens@include\@empty
1835     \let\bbl@ens@exclude\@empty
1836     \def\bbl@ens@fontenc{\relax}%
1837     \def\bbl@tempb##1{%
1838       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1839     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1840     \def\bbl@tempb##1=##2\@{\@namedef{bbl@ens@##1}{##2}}%

```

```

1841 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
1842 \def\bbl@tempc{\bbl@ensure}%
1843 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1844 \expandafter{\bbl@ens@include}}%
1845 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1846 \expandafter{\bbl@ens@exclude}}%
1847 \toks@\expandafter{\bbl@tempc}%
1848 \bbl@exp{%
1849 \endgroup
1850 \def<\bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1851 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1852 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1853 \ifx##1\undefined % 3.32 - Don't assume the macro exists
1854 \edef##1{\noexpand\bbl@nocaption
1855 {\bbl@stripslash##1}{\language\language\bbl@stripslash##1}}%
1856 \fi
1857 \ifx##1\@empty\else
1858 \in@{##1}{#2}%
1859 \ifin@ \else
1860 \bbl@ifunset{\bbl@ensure@\language\language}%
1861 {\bbl@exp{%
1862 \\\DeclareRobustCommand\bbl@ensure@\language[1]{%
1863 \\\foreignlanguage{\language}%
1864 {\ifx\relax#3\else
1865 \\\fontencoding{#3}\selectfont
1866 \fi
1867 #####1}}}%
1868 }%
1869 \toks@\expandafter{##1}%
1870 \edef##1{%
1871 \bbl@csarg\noexpand{ensure@\language}%
1872 {\the\toks@}}%
1873 \fi
1874 \expandafter\bbl@tempb
1875 \fi}%
1876 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1877 \def\bbl@tempa##1{% elt for include list
1878 \ifx##1\@empty\else
1879 \bbl@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
1880 \ifin@ \else
1881 \bbl@tempb##1\@empty
1882 \fi
1883 \expandafter\bbl@tempa
1884 \fi}%
1885 \bbl@tempa#1\@empty}
1886 \def\bbl@captionslist{%
1887 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1888 \contentsname\listfigurename\listtablename\indexname\figurename
1889 \tablename\partname\encname\ccname\headtoname\pagename\seename
1890 \alsoname\proofname\glossaryname}

```

## 9.4 Setting up language files

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

At the start of processing a language definition file we always check the category code of the at-sign. We make sure that it is a ‘letter’ during the processing of the file. We also save its name as the last

called option, even if not loaded.

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

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

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

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

Finally we check \originalTeX.

```
1891 \bbl@trace{Macros for setting language files up}
1892 \def\bbl@ldfinit{%
1893   \let\bbl@screset@empty
1894   \let\BabelStrings\bbl@opt@string
1895   \let\BabelOptions@empty
1896   \let\BabelLanguages\relax
1897   \ifx\originalTeX\@undefined
1898     \let\originalTeX@empty
1899   \else
1900     \originalTeX
1901   \fi}
1902 \def\LdfInit#1#2{%
1903   \chardef\atcatcode=\catcode` \@
1904   \catcode`\@=11\relax
1905   \chardef\eqcatcode=\catcode`\ =
1906   \catcode`\==12\relax
1907   \expandafter\if\expandafter\@backslashchar
1908     \expandafter\@car\string#2\@nil
1909     \ifx#2\@undefined\else
1910       \ldf@quit{#1}%
1911     \fi
1912   \else
1913     \expandafter\ifx\csname#2\endcsname\relax\else
1914       \ldf@quit{#1}%
1915     \fi
1916   \fi
1917   \bbl@ldfinit}
```

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

```
1918 \def\ldf@quit#1{%
1919   \expandafter\main@language\expandafter{#1}%
1920   \catcode`\@=\atcatcode \let\atcatcode\relax
1921   \catcode`\==\eqcatcode \let\eqcatcode\relax
1922   \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.

```
1923 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1924   \bbl@afterlang
1925   \let\bbl@afterlang\relax
1926   \let\BabelModifiers\relax
1927   \let\bbl@screset\relax}%
1928 \def\ldf@finish#1{%
```

```

1929 \ifx\loadlocalcfg\@undefined\else % For LaTeX 209
1930   \loadlocalcfg{#1}%
1931 \fi
1932 \bbl@afterldf{#1}%
1933 \expandafter\main@language\expandafter{#1}%
1934 \catcode`\@=\atcatcode \let\atcatcode\relax
1935 \catcode`\==\eqcatcode \let\eqcatcode\relax}

```

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

```

1936 \@onlypreamble\LdfInit
1937 \@onlypreamble\ldf@quit
1938 \@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.

```

1939 \def\main@language#1{%
1940   \def\bbl@main@language{#1}%
1941   \let\language\main@language % TODO. Set localename
1942   \bbl@id@assign
1943   \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`.

```

1944 \def\bbl@beforestart{%
1945   \bbl@usehooks{beforestart}}}%
1946 \global\let\bbl@beforestart\relax}
1947 \AtBeginDocument{%
1948   \@nameuse{bbl@beforestart}%
1949   \if@filesw
1950     \providecommand\babel@aux[2]{}%
1951     \immediate\write\@mainaux{%
1952       \string\providecommand\string\babel@aux[2]{}%
1953       \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}}%
1954   \fi
1955   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1956   \ifbbl@single % must go after the line above.
1957     \renewcommand\selectlanguage[1]{}%
1958     \renewcommand\foreignlanguage[2]{#2}%
1959     \global\let\babel@aux\@gobbletwo % Also as flag
1960   \fi
1961   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1962 \def\select@language@x#1{%
1963   \ifcase\bbl@select@type
1964     \bbl@ifsamestring\language{#1}{\select@language{#1}}%
1965   \else
1966     \select@language{#1}%
1967   \fi}

```

## 9.5 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.

```

1968 \bbl@trace{Shorhands}
1969 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1970 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1971 \bbl@ifunset{@sanitize}{\bbl@add@sanitize{\@makeother#1}}%
1972 \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1973 \begingroup
1974 \catcode`#1\active
1975 \nfss@catcodes
1976 \ifnum\catcode`#1=\active
1977 \endgroup
1978 \bbl@add\nfss@catcodes{\@makeother#1}%
1979 \else
1980 \endgroup
1981 \fi
1982 \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.

```

1983 \def\bbl@remove@special#1{%
1984 \begingroup
1985 \def\x##1##2{\ifnum`#1=##2\noexpand\@empty
1986 \else\noexpand##1\noexpand##2\fi}%
1987 \def\do{\x\do}%
1988 \def\@makeother{\x\@makeother}%
1989 \edef\x{\endgroup
1990 \def\noexpand\dospecials{\dospecials}%
1991 \expandafter\ifx\csname @sanitize\endcsname\relax\else
1992 \def\noexpand\@sanitize{\@sanitize}%
1993 \fi}%
1994 \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).

```

1995 \def\bbl@active@def#1#2#3#4{%
1996 \namedef{#3#1}{%
1997 \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1998 \bbl@afterelse\bbl@sh@select#2#1{#3#arg#1}{#4#1}%
1999 \else
2000 \bbl@afterfi\csname#2@sh@#1\endcsname
2001 \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.

```

2002 \long\@namedef{#3@arg#1}##1{%
2003   \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
2004     \bbl@afterelse\csname#4#1\endcsname##1%
2005   \else
2006     \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
2007   \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.

```

2008 \def\initiate@active@char#1{%
2009   \bbl@ifunset{active@char\string#1}%
2010   {\bbl@withactive
2011     {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
2012   {}}

```

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).

```

2013 \def\@initiate@active@char#1#2#3{%
2014   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
2015   \ifx#1\@undefined
2016     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
2017   \else
2018     \bbl@csarg\let{oridef@#2}#1%
2019     \bbl@csarg\edef{oridef@#2}{%
2020       \let\noexpand#1%
2021       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
2022   \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*").

```

2023 \ifx#1#3\relax
2024   \expandafter\let\csname normal@char#2\endcsname#3%
2025 \else
2026   \bbl@info{Making #2 an active character}%
2027   \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
2028   \@namedef{normal@char#2}{%
2029     \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
2030   \else
2031     \@namedef{normal@char#2}{#3}%
2032   \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).

```

2033 \bbl@restoreactive{#2}%
2034 \AtBeginDocument{%
2035   \catcode`#2\active
2036   \if@filesw
2037     \immediate\write\@mainaux{\catcode`\string#2\active}%
2038   \fi}%

```

```

2039 \expandafter\bb1@add@special\csname#2\endcsname
2040 \catcode`#2\active
2041 \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⟩`).

```

2042 \let\bb1@tempa\@firstoftwo
2043 \if\string^#2%
2044 \def\bb1@tempa{\noexpand\textormath}%
2045 \else
2046 \ifx\bb1@mathnormal\@undefined\else
2047 \let\bb1@tempa\bb1@mathnormal
2048 \fi
2049 \fi
2050 \expandafter\edef\csname active@char#2\endcsname{%
2051 \bb1@tempa
2052 {\noexpand\if@safe@actives
2053 \noexpand\expandafter
2054 \expandafter\noexpand\csname normal@char#2\endcsname
2055 \noexpand\else
2056 \noexpand\expandafter
2057 \expandafter\noexpand\csname bbl@doactive#2\endcsname
2058 \noexpand\fi}%
2059 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
2060 \bb1@csarg\edef{doactive#2}%
2061 \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!).

```

2062 \bb1@csarg\edef{active@#2}{%
2063 \noexpand\active@prefix\noexpand#1%
2064 \expandafter\noexpand\csname active@char#2\endcsname}%
2065 \bb1@csarg\edef{normal@#2}{%
2066 \noexpand\active@prefix\noexpand#1%
2067 \expandafter\noexpand\csname normal@char#2\endcsname}%
2068 \expandafter\let\expandafter#1\csname bbl@normal@#2\endcsname

```

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.

```

2069 \bb1@active@def#2\user@group{user@active}{language@active}%
2070 \bb1@active@def#2\language@group{language@active}{system@active}%
2071 \bb1@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.

```

2072 \expandafter\edef\csname\user@group @sh#2@@\endcsname
2073 {\expandafter\noexpand\csname normal@char#2\endcsname}%
2074 \expandafter\edef\csname\user@group @sh#2@\string\protect@\endcsname
2075 {\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.

```
2076 \if\string'#2%
2077 \let\prim@s\bbl@prim@s
2078 \let\active@math@prime#1%
2079 \fi
2080 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}
```

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

```
2081 <<{*More package options}>> ≡
2082 \DeclareOption{math=active}{}
2083 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
2084 <</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.

```
2085 \@ifpackagewith{babel}{KeepShorthandsActive}%
2086 {\let\bbl@restoreactive\@gobble}%
2087 {\def\bbl@restoreactive#1{%
2088   \bbl@exp{%
2089     \\\AfterBabelLanguage\\CurrentOption
2090     {\catcode`#1=\the\catcode`#1\relax}%
2091     \\\AtEndOfPackage
2092     {\catcode`#1=\the\catcode`#1\relax}}}%
2093   \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.

```
2094 \def\bbl@sh@select#1#2{%
2095   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
2096     \bbl@afterelse\bbl@scndcs
2097   \else
2098     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
2099   \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.

```
2100 \begingroup
2101 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
2102 {\gdef\active@prefix#1{%
2103   \ifx\protect\@typeset@protect
2104   \else
2105     \ifx\protect\unexpandable@protect
2106       \noexpand#1%
2107     \else
2108       \protect#1%
2109     \fi
2110   \fi}}
```

```

2110     \expandafter\@gobble
2111   \fi}}
2112 {\gdef\active@prefix#1{%
2113   \ifincsname
2114     \string#1%
2115     \expandafter\@gobble
2116   \else
2117     \ifx\protect\@typeset@protect
2118     \else
2119       \ifx\protect\@unexpandable@protect
2120         \noexpand#1%
2121       \else
2122         \protect#1%
2123       \fi
2124     \expandafter\expandafter\expandafter\@gobble
2125   \fi
2126 \fi}}
2127 \endgroup

```

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

```

2128 \newif\if@safe@actives
2129 \@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.

```

2130 \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`.

```

2131 \chardef\bbl@activated\z@
2132 \def\bbl@activate#1{%
2133   \chardef\bbl@activated\@ne
2134   \bbl@withactive{\expandafter\let\expandafter}#1%
2135   \csname bbl@active@\string#1\endcsname}
2136 \def\bbl@deactivate#1{%
2137   \chardef\bbl@activated\tw@
2138   \bbl@withactive{\expandafter\let\expandafter}#1%
2139   \csname bbl@normal@\string#1\endcsname}

```

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

```

\bbl@scndcs
2140 \def\bbl@firstcs#1#2{\csname#1\endcsname}
2141 \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.

```

2142 \def\babel@texpdf#1#2#3#4{%
2143   \ifx\texorpdfstring\undefined
2144     \textormath{#1}{#3}%
2145   \else
2146     \texorpdfstring{\textormath{#1}{#3}}{#2}%
2147     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
2148   \fi}
2149 %
2150 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
2151 \def\@decl@short#1#2#3\@nil#4{%
2152   \def\bbl@tempa{#3}%
2153   \ifx\bbl@tempa\@empty
2154     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
2155     \bbl@ifunset{#1@sh@\string#2@}{}%
2156     {\def\bbl@tempa{#4}%
2157      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
2158      \else
2159        \bbl@info
2160          {Redefining #1 shorthand \string#2\%
2161           in language \CurrentOption}%
2162      \fi}%
2163     \@namedef{#1@sh@\string#2@}{#4}%
2164   \else
2165     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
2166     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
2167     {\def\bbl@tempa{#4}%
2168      \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
2169      \else
2170        \bbl@info
2171          {Redefining #1 shorthand \string#2\string#3\%
2172           in language \CurrentOption}%
2173      \fi}%
2174     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2175   \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.

```

2176 \def\textormath{%
2177   \ifmmode
2178     \expandafter\@secondoftwo
2179   \else
2180     \expandafter\@firstoftwo
2181   \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 group ‘english’ and have a system group called ‘system’.

```

2182 \def\user@group{user}
2183 \def\language@group{english} % TODO. I don't like defaults
2184 \def\system@group{system}

```

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

```

2185 \def\useshorthands{%
2186   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}
2187 \def\bbl@usesh@s#1{%
2188   \bbl@usesh@x

```

```

2189     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
2190     {#1}}
2191 \def\bbl@usesesh@x#1#2{%
2192   \bbl@ifshorthand{#2}%
2193   {\def\user@group{user}%
2194     \initiate@active@char{#2}%
2195     #1%
2196     \bbl@activate{#2}}%
2197   {\bbl@error
2198     {I can't declare a shorthand turned off (\string#2)}
2199     {Sorry, but you can't use shorthands which have been\\%
2200       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.

```

2201 \def\user@language@group{user@\language@group}
2202 \def\bbl@set@user@generic#1#2{%
2203   \bbl@ifunset{user@generic@active#1}%
2204   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
2205     \bbl@active@def#1\user@group{user@generic@active}{language@active}%
2206     \expandafter\edef\csname#2@sh@#1@@\endcsname{%
2207       \expandafter\noexpand\csname normal@char#1\endcsname}%
2208     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
2209       \expandafter\noexpand\csname user@active#1\endcsname}}%
2210   \@empty}
2211 \newcommand\defineshorthand[3][user]{%
2212   \edef\bbl@tempa{\zap@space#1 \@empty}%
2213   \bbl@for\bbl@tempb\bbl@tempa{%
2214     \if*\expandafter\@car\bbl@tempb\@nil
2215       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
2216       \@expandtwoargs
2217       \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
2218     \fi
2219     \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].

```

2220 \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`".

```

2221 \def\aliasshorthand#1#2{%
2222   \bbl@ifshorthand{#2}%
2223   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2224     \ifx\document\@notprerr
2225       \@notshorthand{#2}%
2226     \else
2227       \initiate@active@char{#2}%
2228       \expandafter\let\csname active@char\string#2\endcsname
2229         \csname active@char\string#1\endcsname
2230       \expandafter\let\csname normal@char\string#2\endcsname
2231         \csname normal@char\string#1\endcsname
2232       \bbl@activate{#2}%

```

```

2233     \fi
2234     \fi}%
2235     {\bbl@error
2236       {Cannot declare a shorthand turned off (\string#2)}
2237       {Sorry, but you cannot use shorthands which have been\\%
2238         turned off in the package options}}}

```

\@notshorthand

```

2239 \def\@notshorthand#1{%
2240   \bbl@error{%
2241     The character '\string #1' should be made a shorthand character;\\%
2242     add the command \string\usesshorthands\string{#1\string} to
2243     the preamble.\\%
2244     I will ignore your instruction}%
2245   {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.

```

2246 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2247 \DeclareRobustCommand*\shorthandoff{%
2248   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2249 \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.

```

2250 \def\bbl@switch@sh#1#2{%
2251   \ifx#2\@nnil\else
2252     \bbl@ifunset{\bbl@active@\string#2}%
2253     {\bbl@error
2254       {I can't switch '\string#2' on or off--not a shorthand}%
2255       {This character is not a shorthand. Maybe you made\\%
2256         a typing mistake? I will ignore your instruction.}}%
2257     {\ifcase#1%   off, on, off*
2258       \catcode`#212\relax
2259       \or
2260       \catcode`#2\active
2261       \bbl@ifunset{\bbl@shdef@\string#2}%
2262       {}%
2263       {\bbl@withactive{\expandafter\let\expandafter}#2%
2264         \csname bbl@shdef@\string#2\endcsname
2265         \bbl@csarg\let{\shdef@\string#2}\relax}%
2266       \ifcase\bbl@activated\or
2267         \bbl@activate{#2}%
2268       \else
2269         \bbl@deactivate{#2}%
2270       \fi
2271       \or
2272       \bbl@ifunset{\bbl@shdef@\string#2}%
2273       {\bbl@withactive{\bbl@csarg\let{\shdef@\string#2}}#2}%
2274       {}%
2275       \csname bbl@oricat@\string#2\endcsname
2276       \csname bbl@oridef@\string#2\endcsname
2277     \fi}%

```

```

2278 \bbl@afterfi\bbl@switch@sh#1%
2279 \fi}

```

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

```

2280 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
2281 \def\bbl@putsh#1{%
2282 \bbl@ifunset{bbl@active@\string#1}%
2283 {\bbl@putsh@i#1\@empty\@nnil}%
2284 {\csname bbl@active@\string#1\endcsname}}
2285 \def\bbl@putsh@i#1#2\@nnil{%
2286 \csname\language@group @sh@\string#1@%
2287 \ifx\@empty#2\else\string#2\fi\endcsname}
2288 \ifx\bbl@opt@shorthands\@nnil\else
2289 \let\bbl@s@initiate@active@char\initiate@active@char
2290 \def\initiate@active@char#1{%
2291 \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
2292 \let\bbl@s@switch@sh\bbl@switch@sh
2293 \def\bbl@switch@sh#1#2{%
2294 \ifx#2\@nnil\else
2295 \bbl@afterfi
2296 \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
2297 \fi}
2298 \let\bbl@s@activate\bbl@activate
2299 \def\bbl@activate#1{%
2300 \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
2301 \let\bbl@s@deactivate\bbl@deactivate
2302 \def\bbl@deactivate#1{%
2303 \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
2304 \fi

```

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

```

2305 \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.

```

2306 \def\bbl@prim@s{%
2307 \prime\futurelet\@let@token\bbl@pr@m@s}
2308 \def\bbl@if@primes#1#2{%
2309 \ifx#1\@let@token
2310 \expandafter\@firstoftwo
2311 \else\ifx#2\@let@token
2312 \bbl@afterelse\expandafter\@firstoftwo
2313 \else
2314 \bbl@afterfi\expandafter\@secondoftwo
2315 \fi\fi}
2316 \begingroup
2317 \catcode`\^=7 \catcode`\*= \active \lccode`\*=`^
2318 \catcode`\'=12 \catcode`\`= \active \lccode`\`= ` '
2319 \lowercase{%
2320 \gdef\bbl@pr@m@s{%
2321 \bbl@if@primes" "%
2322 \pr@@@s
2323 {\bbl@if@primes*^\pr@@@t\egroup}}
2324 \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).

```
2325 \initiate@active@char{~}
2326 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2327 \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.

```
2328 \expandafter\def\csname OT1dqpos\endcsname{127}
2329 \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

```
2330 \ifx\f@encoding\@undefined
2331   \def\f@encoding{OT1}
2332 \fi
```

## 9.6 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.

```
2333 \bbl@trace{Language attributes}
2334 \newcommand\languageattribute[2]{%
2335   \def\bbl@tempc{#1}%
2336   \bbl@fixname\bbl@tempc
2337   \bbl@iflanguage\bbl@tempc{%
2338     \bbl@vforeach{#2}{%
```

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

```
2339     \ifx\bbl@known@attribs\@undefined
2340       \in@false
2341     \else
2342       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
2343     \fi
2344     \ifin@
2345       \bbl@warning{%
2346         You have more than once selected the attribute '##1'\%
2347         for language #1. Reported}%
2348     \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.

```
2349       \bbl@exp{%
2350         \\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
2351       \edef\bbl@tempa{\bbl@tempc-##1}%
2352       \expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes%
2353       {\csname\bbl@tempc @attr##1\endcsname}%
2354       {\@attrerr{\bbl@tempc}{##1}}%
2355     \fi}}
2356 \@onlypreamble\languageattribute
```

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

```
2357 \newcommand*{\@attrerr}[2]{%
2358   \bbl@error
2359   {The attribute #2 is unknown for language #1.}%
2360   {Your command will be ignored, type <return> to proceed}}
```

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

```
2361 \def\bbl@declare@ttribute#1#2#3{%
2362   \bbl@xin@{,#2,}{,\BabelModifiers,}%
2363   \ifin@
2364     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2365   \fi
2366   \bbl@add@list\bbl@attributes{#1-#2}%
2367   \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.

```
2368 \def\bbl@ifattributeset#1#2#3#4{%
2369   \ifx\bbl@known@attribs\@undefined
2370     \in@false
2371   \else
2372     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
2373   \fi
2374   \ifin@
2375     \bbl@afterelse#3%
2376   \else
2377     \bbl@afterfi#4%
2378   \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.

```
2379 \def\bbl@ifknown@ttrib#1#2{%
2380   \let\bbl@tempa\@secondoftwo
2381   \bbl@loopx\bbl@tempb{#2}{%
2382     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
2383   \ifin@
2384     \let\bbl@tempa\@firstoftwo
2385   \else
2386   \fi}%
2387   \bbl@tempa}
```

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

```
2388 \def\bbl@clear@ttribs{%
2389   \ifx\bbl@attributes\@undefined\else
2390     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
2391       \expandafter\bbl@clear@ttrib\bbl@tempa.
2392     }%
2393   \let\bbl@attributes\@undefined
```



```

2394 \fi}
2395 \def\bbl@clear@ttrib#1-#2.{%
2396 \expandafter\let\csname#1@attr@#2\endcsname\undefined}
2397 \AtBeginDocument{\bbl@clear@ttribs}

```

## 9.7 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`

```

2398 \bbl@trace{Macros for saving definitions}
2399 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

2400 \newcount\babel@savecnt
2401 \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.

```

2402 \def\babel@save#1{%
2403 \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
2404 \toks@\expandafter{\originalTeX\let#1=}%
2405 \bbl@exp{%
2406 \def\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
2407 \advance\babel@savecnt\@ne}
2408 \def\babel@savevariable#1{%
2409 \toks@\expandafter{\originalTeX #1=}%
2410 \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.

```

2411 \def\bbl@frenchspacing{%
2412 \ifnum\the\sfcodes\@m
2413 \let\bbl@nonfrenchspacing\relax
2414 \else
2415 \frenchspacing
2416 \let\bbl@nonfrenchspacing\nonfrenchspacing
2417 \fi}
2418 \let\bbl@nonfrenchspacing\nonfrenchspacing
2419 \let\bbl@elt\relax
2420 \edef\bbl@fs@chars{%
2421 \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
2422 \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
2423 \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
2424 \def\bbl@pre@fs{%

```

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

```

2425 \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
2426 \edef\bbl@save@sfcodes{\bbl@fs@chars}}%
2427 \def\bbl@post@fs{%
2428   \bbl@save@sfcodes
2429   \edef\bbl@tempa{\bbl@cl{frspc}}}%
2430 \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
2431 \if u\bbl@tempa      % do nothing
2432 \else\if n\bbl@tempa % non french
2433   \def\bbl@elt##1##2##3{%
2434     \ifnum\sfcode`##1=##2\relax
2435       \babel@savevariable{\sfcode`##1}%
2436       \sfcode`##1=##3\relax
2437     \fi}%
2438   \bbl@fs@chars
2439 \else\if y\bbl@tempa % french
2440   \def\bbl@elt##1##2##3{%
2441     \ifnum\sfcode`##1=##3\relax
2442       \babel@savevariable{\sfcode`##1}%
2443       \sfcode`##1=##2\relax
2444     \fi}%
2445   \bbl@fs@chars
2446   \fi\fi\fi}

```

## 9.8 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.

```

2447 \bbl@trace{Short tags}
2448 \def\babeltags#1{%
2449   \edef\bbl@tempa{\zap@space#1 \@empty}%
2450   \def\bbl@tempb##1=##2\@{%
2451     \edef\bbl@tempc{%
2452       \noexpand\newcommand
2453       \expandafter\noexpand\csname ##1\endcsname{%
2454         \noexpand\protect
2455         \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
2456       \noexpand\newcommand
2457       \expandafter\noexpand\csname text##1\endcsname{%
2458         \noexpand\foreignlanguage{##2}}
2459     \bbl@tempc}%
2460   \bbl@for\bbl@tempa\bbl@tempa{%
2461     \expandafter\bbl@tempb\bbl@tempa\@{}}

```

## 9.9 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.

```

2462 \bbl@trace{Hyphens}
2463 \@onlypreamble\babelhyphenation
2464 \AtEndOfPackage{%
2465   \newcommand\babelhyphenation[2][\@empty]{%
2466     \ifx\bbl@hyphenation@relax
2467       \let\bbl@hyphenation@\@empty
2468     \fi
2469     \ifx\bbl@hyphlist\@empty\else
2470       \bbl@warning{%

```

```

2471         You must not intermingle \string\selectlanguage\space and\\%
2472         \string\babelhyphenation\space or some exceptions will not\\%
2473         be taken into account. Reported}%
2474     \fi
2475     \ifx\@empty#1%
2476         \protected@edef\bb1@hyphenation@{\bb1@hyphenation@\space#2}%
2477     \else
2478         \bb1@vforeach{#1}{%
2479             \def\bb1@tempa{##1}%
2480             \bb1@fixname\bb1@tempa
2481             \bb1@iflanguage\bb1@tempa{%
2482                 \bb1@csarg\protected@edef{hyphenation@\bb1@tempa}{%
2483                     \bb1@ifunset{bb1@hyphenation@\bb1@tempa}%
2484                     {}%
2485                     {\csname bb1@hyphenation@\bb1@tempa\endcsname\space}%
2486                     #2}}}%
2487     \fi}}

```

`\bb1@allowhyphens` This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak \hskip 0pt plus 0pt`<sup>32</sup>.

```

2488 \def\bb1@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
2489 \def\bb1@t@one{T1}
2490 \def\allowhyphens{\ifx\cf@encoding\bb1@t@one\else\bb1@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`.

```

2491 \newcommand\babelnullhyphen{\char\hyphenchar\font}
2492 \def\babelhyphen{\active@prefix\babelhyphen\bb1@hyphen}
2493 \def\bb1@hyphen{%
2494     \@ifstar{\bb1@hyphen@i @}{\bb1@hyphen@i \@empty}}
2495 \def\bb1@hyphen@i#1#2{%
2496     \bb1@ifunset{bb1@hy@#1#2\@empty}%
2497     {\csname bb1@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
2498     {\csname bb1@hy@#1#2\@empty\endcsname}}

```

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

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

```

2499 \def\bb1@usehyphen#1{%
2500     \leavevmode
2501     \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2502     \nobreak\hskip\z@skip}
2503 \def\bb1@usehyphen#1{%
2504     \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

2505 \def\bb1@hyphenchar{%
2506     \ifnum\hyphenchar\font=\m@ne
2507         \babelnullhyphen
2508     \else
2509         \char\hyphenchar\font
2510     \fi}

```

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

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

```

2511 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2512 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2513 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2514 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2515 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
2516 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
2517 \def\bbl@hy@repeat{%
2518   \bbl@usehyphen{%
2519     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
2520 \def\bbl@hy@repeat{%
2521   \bbl@usehyphen{%
2522     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
2523 \def\bbl@hy@empty{\hskip\z@skip}
2524 \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.

```

2525 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#1}\bbl@allowhyphens}

```

## 9.10 Multiencoding strings

The aim following commands is to provide a common interface for strings in several encodings. They also contains several hooks which can be used by luatex and xetex. The code is organized here with pseudo-guards, so we start with the basic commands.

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

```

2526 \bbl@trace{Multiencoding strings}
2527 \def\bbl@tglobal#1{\global\let#1#1}
2528 \def\bbl@reacatcode#1{% TODO. Used only once?
2529   \@tempcnta="7F
2530   \def\bbl@tempa{%
2531     \ifnum\@tempcnta>"FF\else
2532       \catcode\@tempcnta=#1\relax
2533       \advance\@tempcnta@ne
2534       \expandafter\bbl@tempa
2535     \fi}%
2536   \bbl@tempa}

```

The second one. We need to patch `\@uclclist`, but it is done once and only if `\SetCase` is used or if strings are encoded. The code is far from satisfactory for several reasons, including the fact `\@uclclist` is not a list any more. Therefore a package option is added to ignore it. Instead of gobbling the macro getting the next two elements (usually `\reserved@a`), we pass it as argument to `\bbl@uclc`. The parser is restarted inside `\langle lang\rangle\bbl@uclc` because we do not know how many expansions are necessary (depends on whether strings are encoded). The last part is tricky – when uppercasing, we have:

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

and starts over (and similarly when lowercasing).

```

2537 \@ifpackagewith{babel}{nocase}%
2538   {\let\bbl@patchuclc\relax}%
2539   {\def\bbl@patchuclc{%
2540     \global\let\bbl@patchuclc\relax
2541     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}}%

```

```

2542 \gdef\bbl@uclc##1{%
2543 \let\bbl@encoded\bbl@encoded@uclc
2544 \bbl@ifunset{\language @bbl@uclc}% and resumes it
2545 {##1}%
2546 {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
2547 \csname\language @bbl@uclc\endcsname}%
2548 {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
2549 \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
2550 \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}
2551 <<*More package options>> ≡
2552 \DeclareOption{nocase}{}
2553 <</More package options>>

```

The following package options control the behavior of \SetString.

```

2554 <<*More package options>> ≡
2555 \let\bbl@opt@strings\@nnil % accept strings=value
2556 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2557 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2558 \def\BabelStringsDefault{generic}
2559 <</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.

```

2560 \@onlypreamble\StartBabelCommands
2561 \def\StartBabelCommands{%
2562 \begingroup
2563 \bbl@recatcode{11}%
2564 <<Macros local to BabelCommands>>
2565 \def\bbl@provstring##1##2{%
2566 \providecommand##1{##2}%
2567 \bbl@tglobal##1}%
2568 \global\let\bbl@scafter\@empty
2569 \let\StartBabelCommands\bbl@startcmds
2570 \ifx\BabelLanguages\relax
2571 \let\BabelLanguages\CurrentOption
2572 \fi
2573 \begingroup
2574 \let\bbl@screaset\@nnil % local flag - disable 1st stopcommands
2575 \StartBabelCommands}
2576 \def\bbl@startcmds{%
2577 \ifx\bbl@screaset\@nnil\else
2578 \bbl@usehooks{stopcommands}{}%
2579 \fi
2580 \endgroup
2581 \begingroup
2582 \@ifstar
2583 {\ifx\bbl@opt@strings\@nnil
2584 \let\bbl@opt@strings\BabelStringsDefault
2585 \fi
2586 \bbl@startcmds@i}%
2587 \bbl@startcmds@i}
2588 \def\bbl@startcmds@i#1#2{%
2589 \edef\bbl@L{\zap@space#1 \@empty}%
2590 \edef\bbl@G{\zap@space#2 \@empty}%
2591 \bbl@startcmds@ii}
2592 \let\bbl@startcmds\StartBabelCommands

```

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

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

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

```
2593 \newcommand\bbl@startcmds@ii[1][\@empty]{%
2594   \let\SetString\@gobbletwo
2595   \let\bbl@stringdef\@gobbletwo
2596   \let\AfterBabelCommands\@gobble
2597   \ifx\@empty#1%
2598     \def\bbl@sc@label{generic}%
2599     \def\bbl@encstring##1##2{%
2600       \ProvideTextCommandDefault##1{##2}%
2601       \bbl@tglobal##1%
2602       \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%
2603     \let\bbl@sctest\in@true
2604   \else
2605     \let\bbl@sc@charset\space % <- zapped below
2606     \let\bbl@sc@fontenc\space % <- " "
2607     \def\bbl@tempa##1=##2\@nil{%
2608       \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
2609     \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
2610     \def\bbl@tempa##1 ##2{% space -> comma
2611       ##1%
2612       \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
2613     \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
2614     \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
2615     \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
2616     \def\bbl@encstring##1##2{%
2617       \bbl@foreach\bbl@sc@fontenc{%
2618         \bbl@ifunset{T@####1}%
2619         }%
2620         {\ProvideTextCommand##1{####1}{##2}%
2621         \bbl@tglobal##1%
2622         \expandafter
2623         \bbl@tglobal\csname####1\string##1\endcsname}}}%
2624     \def\bbl@sctest{%
2625       \bbl@xin@{\bbl@opt@strings,}{\bbl@sc@label,\bbl@sc@fontenc,}}%
2626   \fi
2627   \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
2628   \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
2629     \let\AfterBabelCommands\bbl@aftercmds
2630     \let\SetString\bbl@setstring
2631     \let\bbl@stringdef\bbl@encstring
2632   \else % ie, strings=value
2633     \bbl@sctest
2634   \ifin@
2635     \let\AfterBabelCommands\bbl@aftercmds
2636     \let\SetString\bbl@setstring
2637     \let\bbl@stringdef\bbl@provstring
2638   \fi\fi\fi
2639   \bbl@scswitch
2640   \ifx\bbl@G\@empty
2641     \def\SetString##1##2{%
```

```

2642      \bbl@error{Missing group for string \string##1}%
2643      {You must assign strings to some category, typically\\%
2644       captions or extras, but you set none}}%
2645 \fi
2646 \ifx\@empty#1%
2647   \bbl@usehooks{defaultcommands}{}%
2648 \else
2649   \@expandtwoargs
2650   \bbl@usehooks{encodedcommands}{\bbl@sc@charset}\bbl@sc@fontenc}}%
2651 \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).

```

2652 \def\bbl@forlang#1#2{%
2653   \bbl@for#1\bbl@L{%
2654     \bbl@xin@{,#1},{, \BabelLanguages,}%
2655     \ifin@#2\relax\fi}}
2656 \def\bbl@scswitch{%
2657   \bbl@forlang\bbl@tempa{%
2658     \ifx\bbl@G\@empty\else
2659       \ifx\SetString\@gobbletwo\else
2660         \edef\bbl@GL{\bbl@G\bbl@tempa}%
2661         \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
2662         \ifin@\else
2663           \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
2664           \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
2665         \fi
2666       \fi
2667     \fi}}
2668 \AtEndOfPackage{%
2669   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\#2}}}%
2670   \let\bbl@scswitch\relax
2671   \@onlypreamble\EndBabelCommands
2672   \def\EndBabelCommands{%
2673     \bbl@usehooks{stopcommands}{}%
2674     \endgroup
2675     \endgroup
2676     \bbl@scafter}
2677   \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.

```

2678 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
2679   \bbl@forlang\bbl@tempa{%
2680     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2681     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2682     {\bbl@exp{%
2683       \global\bbbl@add\<\bbl@G\bbl@tempa>\bbbl@scset\#1\<\bbl@LC>}}}%

```

```

2684     {}%
2685     \def\BabelString{#2}%
2686     \bbl@usehooks{stringprocess}{}%
2687     \expandafter\bbl@stringdef
2688     \csname\bbl@LC\expandafter\endcsname\expandafter{\BabelString}}

```

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

```

2689 \ifx\bbl@opt@strings\relax
2690   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2691   \bbl@patchuclc
2692   \let\bbl@encoded\relax
2693   \def\bbl@encoded@uclc#1{%
2694     \@inmathwarn#1%
2695     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2696       \expandafter\ifx\csname ?\string#1\endcsname\relax
2697         \TextSymbolUnavailable#1%
2698       \else
2699         \csname ?\string#1\endcsname
2700       \fi
2701     \else
2702       \csname\cf@encoding\string#1\endcsname
2703     \fi}
2704 \else
2705   \def\bbl@scset#1#2{\def#1{#2}}
2706 \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.

```

2707 <<{*Macros local to BabelCommands}>> ≡
2708 \def\SetStringLoop##1##2{%
2709   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2710   \count@\z@
2711   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2712     \advance\count@\@ne
2713     \toks@\expandafter{\bbl@tempa}%
2714     \bbl@exp{%
2715       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2716       \count@=\the\count@\relax}}}%
2717 <</Macros local to BabelCommands>>

```

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

```

2718 \def\bbl@aftercmds#1{%
2719   \toks@\expandafter{\bbl@scafter#1}%
2720   \xdef\bbl@scafter{\the\toks@}}

```

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

```

2721 <<{*Macros local to BabelCommands}>> ≡
2722 \newcommand\SetCase[3][{}{%
2723   \bbl@patchuclc
2724   \bbl@forlang\bbl@tempa{%
2725     \expandafter\bbl@encstring
2726     \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
2727     \expandafter\bbl@encstring

```



```

2728 \csname\bb1@tempa @bb1@uc\endcsname{##2}%
2729 \expandafter\bb1@encstring
2730 \csname\bb1@tempa @bb1@lc\endcsname{##3}}}%
2731 <</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.

```

2732 <<*Macros local to BabelCommands>> ≡
2733 \newcommand\SetHyphenMap[1]{%
2734 \bb1@forlang\bb1@tempa{%
2735 \expandafter\bb1@stringdef
2736 \csname\bb1@tempa @bb1@hyphenmap\endcsname{##1}}}%
2737 <</Macros local to BabelCommands>>

```

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

```

2738 \newcommand\BabelLower[2]{% one to one.
2739 \ifnum\lccode#1=#2\else
2740 \babel@savevariable{\lccode#1}%
2741 \lccode#1=#2\relax
2742 \fi}
2743 \newcommand\BabelLowerMM[4]{% many-to-many
2744 \@tempcnta=#1\relax
2745 \@tempcntb=#4\relax
2746 \def\bb1@tempa{%
2747 \ifnum\@tempcnta>#2\else
2748 \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2749 \advance\@tempcnta#3\relax
2750 \advance\@tempcntb#3\relax
2751 \expandafter\bb1@tempa
2752 \fi}%
2753 \bb1@tempa}
2754 \newcommand\BabelLowerMO[4]{% many-to-one
2755 \@tempcnta=#1\relax
2756 \def\bb1@tempa{%
2757 \ifnum\@tempcnta>#2\else
2758 \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
2759 \advance\@tempcnta#3
2760 \expandafter\bb1@tempa
2761 \fi}%
2762 \bb1@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

2763 <<*More package options>> ≡
2764 \DeclareOption{hyphenmap=off}{\chardef\bb1@opt@hyphenmap\z@}
2765 \DeclareOption{hyphenmap=first}{\chardef\bb1@opt@hyphenmap\@ne}
2766 \DeclareOption{hyphenmap=select}{\chardef\bb1@opt@hyphenmap\tw@}
2767 \DeclareOption{hyphenmap=other}{\chardef\bb1@opt@hyphenmap\thr@@}
2768 \DeclareOption{hyphenmap=other*}{\chardef\bb1@opt@hyphenmap4\relax}
2769 <</More package options>>

```

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

```

2770 \AtEndOfPackage{%
2771 \ifx\bb1@opt@hyphenmap\undefined
2772 \bb1@xin{,}{\bb1@language@opts}%
2773 \chardef\bb1@opt@hyphenmap\ifin4\else\@ne\fi
2774 \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.

```

2775 \newcommand\setlocalecaption{% TODO. Catch typos. What about ensure?
2776 \@ifstar\bbI@setcaption@s\bbI@setcaption@x}
2777 \def\bbI@setcaption@x#1#2#3{% language caption-name string
2778 \bbI@trim@def\bbI@tempa{#2}%
2779 \bbI@xin@{.template}{\bbI@tempa}%
2780 \ifin@
2781 \bbI@ini@captions@template{#3}{#1}%
2782 \else
2783 \edef\bbI@tempd{%
2784 \expandafter\expandafter\expandafter
2785 \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2786 \bbI@xin@
2787 {\expandafter\string\csname #2name\endcsname}%
2788 {\bbI@tempd}%
2789 \ifin@ % Renew caption
2790 \bbI@xin@{\string\bbI@scset}{\bbI@tempd}%
2791 \ifin@
2792 \bbI@exp{%
2793 \\\bbI@ifsamestring{\bbI@tempa}{\language}%
2794 {\\\bbI@scset\<#2name>\<#1#2name>}%
2795 {}}%
2796 \else % Old way converts to new way
2797 \bbI@ifunset{#1#2name}%
2798 {\bbI@exp{%
2799 \\\bbI@add\<captions#1>\def\<#2name>\<#1#2name>}}%
2800 \\\bbI@ifsamestring{\bbI@tempa}{\language}%
2801 {\def\<#2name>\<#1#2name>}}%
2802 {}}%
2803 {}%
2804 \fi
2805 \else
2806 \bbI@xin@{\string\bbI@scset}{\bbI@tempd}% New
2807 \ifin@ % New way
2808 \bbI@exp{%
2809 \\\bbI@add\<captions#1>\\\bbI@scset\<#2name>\<#1#2name>}%
2810 \\\bbI@ifsamestring{\bbI@tempa}{\language}%
2811 {\\\bbI@scset\<#2name>\<#1#2name>}%
2812 {}}%
2813 \else % Old way, but defined in the new way
2814 \bbI@exp{%
2815 \\\bbI@add\<captions#1>\def\<#2name>\<#1#2name>}}%
2816 \\\bbI@ifsamestring{\bbI@tempa}{\language}%
2817 {\def\<#2name>\<#1#2name>}}%
2818 {}}%
2819 \fi%
2820 \fi
2821 \@namedef{#1#2name}{#3}%
2822 \toks@\expandafter{\bbI@captionslist}%
2823 \bbI@exp{\in@{\<#2name>}{\the\toks@}}%
2824 \ifin@\else
2825 \bbI@exp{\\\bbI@add\\bbI@captionslist{\<#2name>}}%
2826 \bbI@toggle\bbI@captionslist
2827 \fi
2828 \fi}
2829 % \def\bbI@setcaption@s#1#2#3{} % TODO. Not yet implemented

```

## 9.11 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.

```
2830 \bbl@trace{Macros related to glyphs}
2831 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2832   \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2833   \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.

```
2834 \def\save@sf@q#1{\leavevmode
2835   \begingroup
2836   \edef\SF{\spacefactor\the\spacefactor}#1\SF
2837   \endgroup}
```

## 9.12 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`.

### 9.12.1 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.

```
2838 \ProvideTextCommand{\quotedblbase}{OT1}{%
2839   \save@sf@q{\set@low@box{\textquotedblright\}%
2840   \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.

```
2841 \ProvideTextCommandDefault{\quotedblbase}{%
2842   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2843 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2844   \save@sf@q{\set@low@box{\textquoteright\}%
2845   \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.

```
2846 \ProvideTextCommandDefault{\quotesinglbase}{%
2847   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2848 \ProvideTextCommand{\guillemetleft}{OT1}{%
2849   \ifmmode
2850     \ll
2851   \else
2852     \save@sf@q{\nobreak
2853       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2854   \fi}
2855 \ProvideTextCommand{\guillemetright}{OT1}{%
2856   \ifmmode
2857     \gg
2858   \else
2859     \save@sf@q{\nobreak
2860       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2861   \fi}
```

```

2862 \ProvideTextCommand{\guillemotleft}{OT1}{%
2863   \ifmmode
2864     \ll
2865   \else
2866     \save@sf@q{\nobreak
2867       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2868   \fi}
2869 \ProvideTextCommand{\guillemotright}{OT1}{%
2870   \ifmmode
2871     \gg
2872   \else
2873     \save@sf@q{\nobreak
2874       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2875   \fi}

```

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

```

2876 \ProvideTextCommandDefault{\guillemetleft}{%
2877   \UseTextSymbol{OT1}{\guillemetleft}}
2878 \ProvideTextCommandDefault{\guillemetright}{%
2879   \UseTextSymbol{OT1}{\guillemetright}}
2880 \ProvideTextCommandDefault{\guillemotleft}{%
2881   \UseTextSymbol{OT1}{\guillemotleft}}
2882 \ProvideTextCommandDefault{\guillemotright}{%
2883   \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2884 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2885   \ifmmode
2886     <%
2887   \else
2888     \save@sf@q{\nobreak
2889       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2890   \fi}
2891 \ProvideTextCommand{\guilsinglright}{OT1}{%
2892   \ifmmode
2893     >%
2894   \else
2895     \save@sf@q{\nobreak
2896       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2897   \fi}

```

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

```

2898 \ProvideTextCommandDefault{\guilsinglleft}{%
2899   \UseTextSymbol{OT1}{\guilsinglleft}}
2900 \ProvideTextCommandDefault{\guilsinglright}{%
2901   \UseTextSymbol{OT1}{\guilsinglright}}

```

### 9.12.2 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.

```

2902 \DeclareTextCommand{\ij}{OT1}{%
2903   i\kern-0.02em\bbl@allowhyphens j}
2904 \DeclareTextCommand{\IJ}{OT1}{%
2905   I\kern-0.02em\bbl@allowhyphens J}
2906 \DeclareTextCommand{\ij}{T1}{\char188}
2907 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```
2908 \ProvideTextCommandDefault{\ij}{%
2909   \UseTextSymbol{OT1}{\ij}}
2910 \ProvideTextCommandDefault{\IJ}{%
2911   \UseTextSymbol{OT1}{\IJ}}
```

`\dj` The croatian language needs the letters `\dj` and `\DJ`; they are available in the T1 encoding, but not in the OT1 encoding by default.

Some code to construct these glyphs for the OT1 encoding was made available to me by Stipčević Mario, (stipcevic@olimp.irb.hr).

```
2912 \def\crrtic@{\hrule height0.1ex width0.3em}
2913 \def\crttic@{\hrule height0.1ex width0.33em}
2914 \def\ddj@{%
2915   \setbox0\hbox{d}\dimen@=\ht0
2916   \advance\dimen@1ex
2917   \dimen@.45\dimen@
2918   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2919   \advance\dimen@ii.5ex
2920   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2921 \def\DDJ@{%
2922   \setbox0\hbox{D}\dimen@=.55\ht0
2923   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2924   \advance\dimen@ii.15ex % correction for the dash position
2925   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2926   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2927   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2928 %
2929 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2930 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2931 \ProvideTextCommandDefault{\dj}{%
2932   \UseTextSymbol{OT1}{\dj}}
2933 \ProvideTextCommandDefault{\DJ}{%
2934   \UseTextSymbol{OT1}{\DJ}}
```

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

```
2935 \DeclareTextCommand{\SS}{OT1}{\SS}
2936 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

### 9.12.3 Shorthands for quotation marks

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

`\glq` The ‘german’ single quotes.

```
\grq
2937 \ProvideTextCommandDefault{\glq}{%
2938   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2939 \ProvideTextCommand{\grq}{T1}{%
2940   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2941 \ProvideTextCommand{\grq}{TU}{%
2942   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2943 \ProvideTextCommand{\grq}{OT1}{%
2944   \save@sf@q{\kern-.0125em
```

```

2945 \textormath{\textquoteleft}{\mbox{\textquoteleft}}%
2946 \kern.07em\relax}}
2947 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}

\glqq The ‘german’ double quotes.
\grqq 2948 \ProvideTextCommandDefault{\glqq}{%
2949 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

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

2950 \ProvideTextCommand{\grqq}{T1}{%
2951 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2952 \ProvideTextCommand{\grqq}{TU}{%
2953 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2954 \ProvideTextCommand{\grqq}{OT1}{%
2955 \save@sf@q{\kern-.07em
2956 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2957 \kern.07em\relax}}
2958 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}

\flq The ‘french’ single guillemets.
\frq 2959 \ProvideTextCommandDefault{\flq}{%
2960 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2961 \ProvideTextCommandDefault{\frq}{%
2962 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

\flqq The ‘french’ double guillemets.
\frqq 2963 \ProvideTextCommandDefault{\flqq}{%
2964 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2965 \ProvideTextCommandDefault{\frqq}{%
2966 \textormath{\guillemetright}{\mbox{\guillemetright}}}

```

### 9.12.4 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).

```

2967 \def\umlauthigh{%
2968 \def\bbl@umlauta##1{\leavevmode\bgroup%
2969 \expandafter\accent\csname\fontencoding dqpos\endcsname
2970 ##1\bbl@allowhyphens\egroup}%
2971 \let\bbl@umlaute\bbl@umlauta}
2972 \def\umlautlow{%
2973 \def\bbl@umlauta{\protect\lower@umlaut}}
2974 \def\umlautelow{%
2975 \def\bbl@umlaute{\protect\lower@umlaut}}
2976 \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.

```

2977 \expandafter\ifx\csname U@D\endcsname\relax
2978 \csname newdimen\endcsname U@D
2979 \fi

```

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

```

2980 \def\lower@umlaut#1{%
2981   \leavevmode\bgroup
2982     \U@D 1ex%
2983     {\setbox\z@\hbox{%
2984       \expandafter\char\csname\fontencoding dqpos\endcsname}%
2985       \dimen@ -.45ex\advance\dimen@\ht\z@
2986       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2987     \expandafter\accent\csname\fontencoding dqpos\endcsname
2988     \fontdimen5\font\U@D #1%
2989   \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).

```

2990 \AtBeginDocument{%
2991   \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
2992   \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
2993   \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{i}}%
2994   \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{i}}%
2995   \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
2996   \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
2997   \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
2998   \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
2999   \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
3000   \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
3001   \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`.

```

3002 \ifx\l@english\undefined
3003   \chardef\l@english\z@
3004 \fi
3005 % The following is used to cancel rules in ini files (see Amharic).
3006 \ifx\l@unhyphenated\undefined
3007   \newlanguage\l@unhyphenated
3008 \fi

```

## 9.13 Layout

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

```

3009 \bbl@trace{Bidi layout}
3010 \providecommand\IfBabelLayout[3]{#3}%
3011 \newcommand\BabelPatchSection[1]{%
3012   \@ifundefined{#1}{}{%
3013     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
3014     \@namedef{#1}{%
3015       \@ifstar{\bbl@presec@s{#1}}%
3016       {\@dblarg{\bbl@presec@x{#1}}}}}%

```

```

3017 \def\bbl@presec@x#1[#2]#3{%
3018   \bbl@exp{%
3019     \\select@language@x{\bbl@main@language}%
3020     \\bbl@cs{sspre@#1}%
3021     \\bbl@cs{ss@#1}%
3022     [\\foreignlanguage{\language}{\unexpanded{#2}}]%
3023     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
3024     \\select@language@x{\language}}%
3025 \def\bbl@presec@s#1#2{%
3026   \bbl@exp{%
3027     \\select@language@x{\bbl@main@language}%
3028     \\bbl@cs{sspre@#1}%
3029     \\bbl@cs{ss@#1}*%
3030     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
3031     \\select@language@x{\language}}%
3032 \IfBabelLayout{sectioning}%
3033   {\BabelPatchSection{part}%
3034    \BabelPatchSection{chapter}%
3035    \BabelPatchSection{section}%
3036    \BabelPatchSection{subsection}%
3037    \BabelPatchSection{subsubsection}%
3038    \BabelPatchSection{paragraph}%
3039    \BabelPatchSection{subparagraph}%
3040   \def\babel@toc#1{%
3041     \select@language@x{\bbl@main@language}}}%
3042 \IfBabelLayout{captions}%
3043   {\BabelPatchSection{caption}}%

```

## 9.14 Load engine specific macros

```

3044 \bbl@trace{Input engine specific macros}
3045 \ifcase\bbl@engine
3046   \input txtbabel.def
3047 \or
3048   \input luababel.def
3049 \or
3050   \input xebabel.def
3051 \fi

```

## 9.15 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.

```

3052 \bbl@trace{Creating languages and reading ini files}
3053 \let\bbl@extend@ini\@gobble
3054 \newcommand\babelprovide[2][{}%
3055   \let\bbl@savelanguage\language
3056   \edef\bbl@savelocaleid{\the\localeid}%
3057   % Set name and locale id
3058   \edef\language{#2}%
3059   \bbl@id@assign
3060   % Initialize keys
3061   \let\bbl@KVP@captions\@nil
3062   \let\bbl@KVP@date\@nil
3063   \let\bbl@KVP@import\@nil
3064   \let\bbl@KVP@main\@nil
3065   \let\bbl@KVP@script\@nil
3066   \let\bbl@KVP@language\@nil

```



```

3067 \let\bb1@KVP@hyphenrules\@nil
3068 \let\bb1@KVP@linebreaking\@nil
3069 \let\bb1@KVP@justification\@nil
3070 \let\bb1@KVP@mapfont\@nil
3071 \let\bb1@KVP@maparabic\@nil
3072 \let\bb1@KVP@mapdigits\@nil
3073 \let\bb1@KVP@intraspace\@nil
3074 \let\bb1@KVP@intrapenalty\@nil
3075 \let\bb1@KVP@onchar\@nil
3076 \let\bb1@KVP@transforms\@nil
3077 \global\let\bb1@release@transforms\@empty
3078 \let\bb1@KVP@alph\@nil
3079 \let\bb1@KVP@Alph\@nil
3080 \let\bb1@KVP@labels\@nil
3081 \bb1@csarg\let{KVP@labels*}\@nil
3082 \global\let\bb1@inidata\@empty
3083 \global\let\bb1@extend@ini\@gobble
3084 \gdef\bb1@key@list{;}%
3085 \bb1@forkv{#1}{% TODO - error handling
3086   \in@{/}{##1}%
3087   \ifin@
3088     \global\let\bb1@extend@ini\bb1@extend@ini@aux
3089     \bb1@renewinikey##1\@{##2}%
3090   \else
3091     \bb1@csarg\def{KVP@##1}{##2}%
3092   \fi}%
3093 \chardef\bb1@howloaded=% 0:none; 1:ldf without ini; 2:ini
3094 \bb1@ifunset{date#2}\z@{\bb1@ifunset{\bb1@llevel@#2}\@ne\tw}%
3095 % == init ==
3096 \ifx\bb1@screset\@undefined
3097   \bb1@ldfinit
3098 \fi
3099 % ==
3100 \let\bb1@lbkflag\relax % \@empty = do setup linebreak
3101 \ifcase\bb1@howloaded
3102   \let\bb1@lbkflag\@empty % new
3103 \else
3104   \ifx\bb1@KVP@hyphenrules\@nil\else
3105     \let\bb1@lbkflag\@empty
3106   \fi
3107   \ifx\bb1@KVP@import\@nil\else
3108     \let\bb1@lbkflag\@empty
3109   \fi
3110 \fi
3111 % == import, captions ==
3112 \ifx\bb1@KVP@import\@nil\else
3113   \bb1@exp{\bb1@ifblank{\bb1@KVP@import}}%
3114   {\ifx\bb1@initload\relax
3115     \begingroup
3116       \def\BabelBeforeIni##1##2{\gdef\bb1@KVP@import{##1}\endinput}%
3117       \bb1@input@texini{##2}%
3118     \endgroup
3119   \else
3120     \xdef\bb1@KVP@import{\bb1@initload}%
3121   \fi}%
3122 {}%
3123 \fi
3124 \ifx\bb1@KVP@captions\@nil
3125   \let\bb1@KVP@captions\bb1@KVP@import

```

```

3126 \fi
3127 % ==
3128 \ifx\bbbl@KVP@transforms\@nil\else
3129 \bbbl@replace\bbbl@KVP@transforms{ }{,}%
3130 \fi
3131 % Load ini
3132 \ifcase\bbbl@howloaded
3133 \bbbl@provide@new{#2}%
3134 \else
3135 \bbbl@ifblank{#1}%
3136 {}% With \bbbl@load@basic below
3137 {\bbbl@provide@renew{#2}}%
3138 \fi
3139 % Post tasks
3140 % -----
3141 % == subsequent calls after the first provide for a locale ==
3142 \ifx\bbbl@inidata\@empty\else
3143 \bbbl@extend@ini{#2}%
3144 \fi
3145 % == ensure captions ==
3146 \ifx\bbbl@KVP@captions\@nil\else
3147 \bbbl@ifunset{bbbl@extracaps@#2}%
3148 {\bbbl@exp{\labelensure[exclude=\\today]{#2}}}%
3149 {\toks@\\expandafter\\expandafter\\expandafter
3150 {\csname bbbl@extracaps@#2\endcsname}%
3151 \bbbl@exp{\labelensure[exclude=\\today,include=\\the\\toks@]}{#2}}%
3152 \bbbl@ifunset{bbbl@ensure@\\language}%
3153 {\bbbl@exp{%
3154 \\\DeclareRobustCommand\<bbbl@ensure@\\language>[1]{%
3155 \\\foreignlanguage{\\language}%
3156 {###1}}}%
3157 }%
3158 \bbbl@exp{%
3159 \\\bbbl@tglobal\<bbbl@ensure@\\language>%
3160 \\\bbbl@tglobal\<bbbl@ensure@\\language\\space>}%
3161 \fi
3162 % ==
3163 % At this point all parameters are defined if 'import'. Now we
3164 % execute some code depending on them. But what about if nothing was
3165 % imported? We just set the basic parameters, but still loading the
3166 % whole ini file.
3167 \bbbl@load@basic{#2}%
3168 % == script, language ==
3169 % Override the values from ini or defines them
3170 \ifx\bbbl@KVP@script\@nil\else
3171 \bbbl@csarg\edef{sname@#2}{\bbbl@KVP@script}%
3172 \fi
3173 \ifx\bbbl@KVP@language\@nil\else
3174 \bbbl@csarg\edef{lname@#2}{\bbbl@KVP@language}%
3175 \fi
3176 % == onchar ==
3177 \ifx\bbbl@KVP@onchar\@nil\else
3178 \bbbl@luahyphenate
3179 \directlua{
3180 if Babel.locale_mapped == nil then
3181 Babel.locale_mapped = true
3182 Babel.linebreaking.add_before(Babel.locale_map)
3183 Babel.loc_to_scr = {}
3184 Babel.chr_to_loc = Babel.chr_to_loc or {}

```

```

3185     end}%
3186 \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
3187 \ifin@
3188   \ifx\bbl@starthyphens\undefined % Needed if no explicit selection
3189     \AddBabelHook{babel-onchar}{beforestart}{{\bbl@starthyphens}}%
3190   \fi
3191   \bbl@exp{\bbl@add\bbl@starthyphens
3192     {\bbl@patterns@lua{\language}}}%
3193   % TODO - error/warning if no script
3194   \directlua{
3195     if Babel.script_blocks['\bbl@cl{sbc}'] then
3196       Babel.loc_to_scr[\the\localeid] =
3197         Babel.script_blocks['\bbl@cl{sbc}']
3198       Babel.locale_props[\the\localeid].lc = \the\localeid\space
3199       Babel.locale_props[\the\localeid].lg = \the\nameuse{l@\language}\space
3200     end
3201   }%
3202 \fi
3203 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
3204 \ifin@
3205   \bbl@ifunset{bbl@lsys@\language}{\bbl@provide@lsys{\language}}{}%
3206   \bbl@ifunset{bbl@wdir@\language}{\bbl@provide@dirs{\language}}{}%
3207   \directlua{
3208     if Babel.script_blocks['\bbl@cl{sbc}'] then
3209       Babel.loc_to_scr[\the\localeid] =
3210         Babel.script_blocks['\bbl@cl{sbc}']
3211     end}%
3212   \ifx\bbl@mapselect\undefined % TODO. almost the same as mapfont
3213     \AtBeginDocument{%
3214       \expandafter\bbl@add\csname selectfont \endcsname{{\bbl@mapselect}}%
3215       {\selectfont}}%
3216     \def\bbl@mapselect{%
3217       \let\bbl@mapselect\relax
3218       \edef\bbl@prefontid{\fontid\font}}%
3219     \def\bbl@mapdir##1{%
3220       {\def\language{##1}%
3221         \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
3222         \bbl@switchfont
3223         \directlua{
3224           Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
3225             [\bbl@prefontid] = \fontid\font\space}}}%
3226     \fi
3227     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
3228   \fi
3229   % TODO - catch non-valid values
3230 \fi
3231 % == mapfont ==
3232 % For bidi texts, to switch the font based on direction
3233 \ifx\bbl@KVP@mapfont\@nil\else
3234   \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
3235   {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\%
3236     mapfont. Use 'direction'.%
3237     {See the manual for details.}}}%
3238   \bbl@ifunset{bbl@lsys@\language}{\bbl@provide@lsys{\language}}{}%
3239   \bbl@ifunset{bbl@wdir@\language}{\bbl@provide@dirs{\language}}{}%
3240   \ifx\bbl@mapselect\undefined % TODO. See onchar
3241     \AtBeginDocument{%
3242       \expandafter\bbl@add\csname selectfont \endcsname{{\bbl@mapselect}}%
3243       {\selectfont}}%

```

```

3244 \def\bb@mapselect{%
3245 \let\bb@mapselect\relax
3246 \edef\bb@prefontid{\fontid\font}}%
3247 \def\bb@mapdir##1{%
3248 {\def\languagename{##1}%
3249 \let\bb@ifrestoring\@firstoftwo % avoid font warning
3250 \bb@switchfont
3251 \directlua{Babel.fontmap
3252 [\the\csname \bb@wdir@##1\endcsname]%
3253 [\bb@prefontid]=\fontid\font}}}%
3254 \fi
3255 \bb@exp{\bb@add\bb@mapselect{\bb@mapdir{\languagename}}}%
3256 \fi
3257 % == Line breaking: intraspace, intrapenalty ==
3258 % For CJK, East Asian, Southeast Asian, if interspace in ini
3259 \ifx\bb@KVP@intraspace\@nil\else % We can override the ini or set
3260 \bb@csarg\edef{intsp@#2}{\bb@KVP@intraspace}%
3261 \fi
3262 \bb@provide@intraspace
3263 % == Line breaking: justification ==
3264 \ifx\bb@KVP@justification\@nil\else
3265 \let\bb@KVP@linebreaking\bb@KVP@justification
3266 \fi
3267 \ifx\bb@KVP@linebreaking\@nil\else
3268 \bb@xin@{,\bb@KVP@linebreaking,}{,elongated,kashida,cjk,unhyphenated,}%
3269 \ifin@
3270 \bb@csarg\xdef
3271 {\lnbrk@\languagename}{\expandafter\@car\bb@KVP@linebreaking\@nil}%
3272 \fi
3273 \fi
3274 \bb@xin@{/e}{/\bb@cl{\lnbrk}}%
3275 \ifin@\else\bb@xin@{/k}{/\bb@cl{\lnbrk}}\fi
3276 \ifin@\bb@arabicjust\fi
3277 % == Line breaking: hyphenate.other.(locale|script) ==
3278 \ifx\bb@lbkflag\@empty
3279 \bb@ifunset{\bb@hyotl@\languagename}{}%
3280 {\bb@csarg\bb@replace{\hyotl@\languagename}{ }{,}%
3281 \bb@startcommands*\languagename}{}%
3282 \bb@csarg\bb@foreach{\hyotl@\languagename}{%
3283 \ifcase\bb@engine
3284 \ifnum##1<257
3285 \SetHyphenMap{\BabelLower{##1}{##1}}%
3286 \fi
3287 \else
3288 \SetHyphenMap{\BabelLower{##1}{##1}}%
3289 \fi}%
3290 \bb@endcommands}%
3291 \bb@ifunset{\bb@hyots@\languagename}{}%
3292 {\bb@csarg\bb@replace{\hyots@\languagename}{ }{,}%
3293 \bb@csarg\bb@foreach{\hyots@\languagename}{%
3294 \ifcase\bb@engine
3295 \ifnum##1<257
3296 \global\lccode##1=##1\relax
3297 \fi
3298 \else
3299 \global\lccode##1=##1\relax
3300 \fi}}}%
3301 \fi
3302 % == Counters: maparabic ==

```

```

3303 % Native digits, if provided in ini (TeX level, xe and lua)
3304 \ifcase\bb1@engine\else
3305   \bb1@ifunset{\bb1@dgnat@\language\name}{}%
3306   {\xexpandafter\xifx\csname \bb1@dgnat@\language\name\endcsname\@empty\else
3307     \xexpandafter\xexpandafter\xexpandafter
3308     \bb1@setdigits\csname \bb1@dgnat@\language\name\endcsname
3309     \ifx\bb1@KVP@maparabic\@nil\else
3310       \ifx\bb1@latinarabic\@undefined
3311         \xexpandafter\let\xexpandafter\@arabic
3312         \csname \bb1@counter@\language\name\endcsname
3313       \else % ie, if layout=counters, which redefines \@arabic
3314         \xexpandafter\let\xexpandafter\bb1@latinarabic
3315         \csname \bb1@counter@\language\name\endcsname
3316       \fi
3317     \fi
3318   \fi}%
3319 \fi
3320 % == Counters: mapdigits ==
3321 % Native digits (lua level).
3322 \ifodd\bb1@engine
3323   \ifx\bb1@KVP@mapdigits\@nil\else
3324     \bb1@ifunset{\bb1@dgnat@\language\name}{}%
3325     {\RequirePackage{luatexbase}%
3326      \bb1@activate@preotf
3327      \directlua{
3328        Babel = Babel or {} %%% -> presets in luababel
3329        Babel.digits_mapped = true
3330        Babel.digits = Babel.digits or {}
3331        Babel.digits[\the\localeid] =
3332          table.pack(string.utfvalue('\bb1@cl{dgnat}'))
3333        if not Babel.numbers then
3334          function Babel.numbers(head)
3335            local LOCALE = luatexbase.registernumber'\bb1@attr@locale'
3336            local GLYPH = node.id'glyph'
3337            local inmath = false
3338            for item in node.traverse(head) do
3339              if not inmath and item.id == GLYPH then
3340                local temp = node.get_attribute(item, LOCALE)
3341                if Babel.digits[temp] then
3342                  local chr = item.char
3343                  if chr > 47 and chr < 58 then
3344                    item.char = Babel.digits[temp][chr-47]
3345                  end
3346                end
3347                elseif item.id == node.id'math' then
3348                  inmath = (item.subtype == 0)
3349                end
3350              end
3351            end
3352            return head
3353          end
3354        }%
3355      \fi
3356    \fi
3357 % == Counters: alph, Alph ==
3358 % What if extras<lang> contains a \babel@save\@alph? It won't be
3359 % restored correctly when exiting the language, so we ignore
3360 % this change with the \bb1@alph@saved trick.
3361 \ifx\bb1@KVP@alph\@nil\else

```

```

3362 \bbl@extras@wrap{\bbl@alph@saved}%
3363 {\let\bbl@alph@saved\@alph}%
3364 {\let\@alph\bbl@alph@saved
3365 \babel@save\@alph}%
3366 \bbl@exp{%
3367 \bbl@add\<extras\language>{%
3368 \let\@alph\bbl@cntr\bbl@KVP@alph @\language>}}%
3369 \fi
3370 \ifx\bbl@KVP@Alph\@nil\else
3371 \bbl@extras@wrap{\bbl@Alph@saved}%
3372 {\let\bbl@Alph@saved\@Alph}%
3373 {\let\@Alph\bbl@Alph@saved
3374 \babel@save\@Alph}%
3375 \bbl@exp{%
3376 \bbl@add\<extras\language>{%
3377 \let\@Alph\bbl@cntr\bbl@KVP@Alph @\language>}}%
3378 \fi
3379 % == require.babel in ini ==
3380 % To load or reload the babel-*.tex, if require.babel in ini
3381 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
3382 \bbl@ifunset\bbl@rqtex\language\}%
3383 {\expandafter\ifx\csname bbl@rqtex\language\endcsname\@empty\else
3384 \let\BabelBeforeIni\@gobbletwo
3385 \chardef\atcatcode=\catcode\@
3386 \catcode\@=11\relax
3387 \bbl@input@texini{\bbl@cs{rqtex\language}}%
3388 \catcode\@=\atcatcode
3389 \let\atcatcode\relax
3390 \global\bbl@csarg\let{rqtex\language}\relax
3391 \fi}%
3392 \fi
3393 % == frenchspacing ==
3394 \ifcase\bbl@howloaded\in@true\else\in@false\fi
3395 \ifin@else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
3396 \ifin@
3397 \bbl@extras@wrap{\bbl@pre@fs}%
3398 {\bbl@pre@fs}%
3399 {\bbl@post@fs}%
3400 \fi
3401 % == Release saved transforms ==
3402 \bbl@release@transforms\relax % \relax closes the last item.
3403 % == main ==
3404 \ifx\bbl@KVP@main\@nil % Restore only if not 'main'
3405 \let\language\bbl@savelangname
3406 \chardef\localeid\bbl@savelocaleid\relax
3407 \fi}

```

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

```

3408 \def\bbl@provide@new#1{%
3409 \namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
3410 \namedef{extras#1}{}%
3411 \namedef{noextras#1}{}%
3412 \bbl@startcommands*{#1}{captions}%
3413 \ifx\bbl@KVP@captions\@nil % and also if import, implicit
3414 \def\bbl@tempb##1{% elt for \bbl@captionslist
3415 \ifx##1\@empty\else
3416 \bbl@exp{%
3417 \SetString\##1%

```

```

3418      \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
3419      \expandafter\bbl@tempb
3420      \fi}%
3421      \expandafter\bbl@tempb\bbl@captionslist\@empty
3422      \else
3423        \ifx\bbl@initoload\relax
3424          \bbl@read@ini{\bbl@KVP@captions}2%   % Here letters cat = 11
3425        \else
3426          \bbl@read@ini{\bbl@initoload}2%       % Same
3427        \fi
3428      \fi
3429      \StartBabelCommands*{#1}{date}%
3430      \ifx\bbl@KVP@import\@nil
3431        \bbl@exp{%
3432          \\SetString\\today{\\bbl@nocaption{today}{#1today}}}%
3433        \else
3434          \bbl@savetoday
3435          \bbl@savedate
3436        \fi
3437      \bbl@endcommands
3438      \bbl@load@basic{#1}%
3439      % == hyphenmins == (only if new)
3440      \bbl@exp{%
3441        \gdef\<#1hyphenmins>{%
3442          {\bbl@ifunset{\bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
3443          {\bbl@ifunset{\bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}%
3444          % == hyphenrules (also in renew) ==
3445          \bbl@provide@hyphens{#1}%
3446          \ifx\bbl@KVP@main\@nil\else
3447            \expandafter\main@language\expandafter{#1}%
3448          \fi}
3449      %
3450      \def\bbl@provide@renew#1{%
3451        \ifx\bbl@KVP@captions\@nil\else
3452          \StartBabelCommands*{#1}{captions}%
3453          \bbl@read@ini{\bbl@KVP@captions}2%   % Here all letters cat = 11
3454          \EndBabelCommands
3455        \fi
3456        \ifx\bbl@KVP@import\@nil\else
3457          \StartBabelCommands*{#1}{date}%
3458          \bbl@savetoday
3459          \bbl@savedate
3460          \EndBabelCommands
3461        \fi
3462        % == hyphenrules (also in new) ==
3463        \ifx\bbl@lbkflag\@empty
3464          \bbl@provide@hyphens{#1}%
3465        \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.)

```

3466 \def\bbl@load@basic#1{%
3467   \ifcase\bbl@howloaded\else
3468     \ifcase0\csname bbl@llevel\@languagename\endcsname
3469       \bbl@csarg\let{lname@languagename}\relax
3470     \fi
3471   \fi
3472   \bbl@ifunset{\bbl@lname@#1}%

```

```

3473 {\def\BabelBeforeIni##1##2{%
3474   \begingroup
3475   \let\bbl@ini@captions@aux\@gobbletwo
3476   \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6{%
3477     \bbl@read@ini{##1}1%
3478     \ifx\bbl@initoload\relax\endinput\fi
3479   \endgroup}%
3480   \begingroup      % boxed, to avoid extra spaces:
3481   \ifx\bbl@initoload\relax
3482     \bbl@input@texini{##1}%
3483   \else
3484     \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
3485     \fi
3486   \endgroup}%
3487   {}}

```

The hyphenrules option is handled with an auxiliary macro.

```

3488 \def\bbl@provide@hyphens#1{%
3489   \let\bbl@tempa\relax
3490   \ifx\bbl@KVP@hyphenrules\@nil\else
3491     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
3492     \bbl@foreach\bbl@KVP@hyphenrules{%
3493       \ifx\bbl@tempa\relax      % if not yet found
3494         \bbl@ifsamestring{##1}{+}%
3495         {{\bbl@exp{\addlanguage\<l@##1>}}}%
3496         {}}%
3497         \bbl@ifunset{l@##1}%
3498         {}%
3499         {\bbl@exp{\let\bbl@tempa\<l@##1>}}%
3500       \fi}%
3501   \fi
3502   \ifx\bbl@tempa\relax %      if no opt or no language in opt found
3503     \ifx\bbl@KVP@import\@nil
3504       \ifx\bbl@initoload\relax\else
3505         \bbl@exp{%
3506           and hyphenrules is not empty
3507           \\bbl@ifblank{\bbl@cs{hyphr@#1}}%
3508           {}}%
3509           {\let\\bbl@tempa\<l@bbl@cl{hyphr}>}}%
3510       \fi
3511     \else % if importing
3512       \bbl@exp{%
3513         and hyphenrules is not empty
3514         \\bbl@ifblank{\bbl@cs{hyphr@#1}}%
3515         {}}%
3516       {\let\\bbl@tempa\<l@bbl@cl{hyphr}>}}%
3517     \fi
3518   \fi
3519   \bbl@ifunset{\bbl@tempa}%      ie, relax or undefined
3520   {\bbl@ifunset{l@#1}%          no hyphenrules found - fallback
3521     {\bbl@exp{\adddialect\<l@#1>\language}}%
3522     {}}%
3523     so, l@<lang> is ok - nothing to do
3524     {\bbl@exp{\adddialect\<l@#1>\bbl@tempa}}% found in opt list or ini

```

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

```

3522 \def\bbl@input@texini#1{%
3523   \bbl@bsphack
3524   \bbl@exp{%
3525     \catcode`\%%=14 \catcode`\==0
3526     \catcode`\{=1 \catcode`\}=2
3527     \lowercase{\InputIfFileExists{babel-#1.tex}{}}%

```



```

3528 \catcode`\\%=\the\catcode`\%\relax
3529 \catcode`\\=\the\catcode`\%\relax
3530 \catcode`\\={\the\catcode`\{\relax
3531 \catcode`\\}=\the\catcode`\}\relax}%
3532 \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.

```

3533 \def\bbl@inline#1\bbl@inline{%
3534 \@ifnextchar[\bbl@iniset{\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
3535 \def\bbl@iniset[#1]#2\@@{\def\bbl@section{#1}}
3536 \def\bbl@iniskip#1\@@{% if starts with ;
3537 \def\bbl@inistore#1=#2\@@{% full (default)
3538 \bbl@trim@def\bbl@tempa{#1}%
3539 \bbl@trim\toks@{#2}%
3540 \bbl@xin@{\bbl@section/\bbl@tempa;}{\bbl@key@list}%
3541 \ifin@else
3542 \bbl@exp{%
3543 \\g@addto@macro\\bbl@inidata{%
3544 \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
3545 \fi}
3546 \def\bbl@inistore@min#1=#2\@@{% minimal (maybe set in \bbl@read@ini)
3547 \bbl@trim@def\bbl@tempa{#1}%
3548 \bbl@trim\toks@{#2}%
3549 \bbl@xin@{.identification.}{.\bbl@section.}%
3550 \ifin@
3551 \bbl@exp{\\g@addto@macro\\bbl@inidata{%
3552 \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
3553 \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.

```

3554 \ifx\bbl@readstream\undefined
3555 \csname newread\endcsname\bbl@readstream
3556 \fi
3557 \def\bbl@read@ini#1#2{%
3558 \global\let\bbl@extend@ini@gobble
3559 \openin\bbl@readstream=babel-#1.ini
3560 \ifeof\bbl@readstream
3561 \bbl@error
3562 {There is no ini file for the requested language\\%
3563 (#1). Perhaps you misspelled it or your installation\\%
3564 is not complete.}%
3565 {Fix the name or reinstall babel.}%
3566 \else
3567 % == Store ini data in \bbl@inidata ==
3568 \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
3569 \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
3570 \bbl@info{Importing
3571 \ifcase#2font and identification \or basic \fi
3572 data for \languagename\\%
3573 from babel-#1.ini. Reported}%
3574 \ifnum#2=\z@
3575 \global\let\bbl@inidata\empty

```

```

3576     \let\bbl@inistore\bbl@inistore@min    % Remember it's local
3577 \fi
3578 \def\bbl@section{identification}%
3579 \bbl@exp{\bbl@inistore tag.ini=#1\\@@}%
3580 \bbl@inistore load.level=#2\\@@
3581 \loop
3582 \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3583     \endlinechar\m@ne
3584     \read\bbl@readstream to \bbl@line
3585     \endlinechar`^^M
3586     \ifx\bbl@line\empty\else
3587         \expandafter\bbl@inline\bbl@line\bbl@inline
3588     \fi
3589 \repeat
3590 % == Process stored data ==
3591 \bbl@csarg\def{lini@language}{#1}%
3592 \bbl@read@ini@aux
3593 % == 'Export' data ==
3594 \bbl@ini@exports{#2}%
3595 \global\bbl@csarg\let{inidata@language}\bbl@inidata
3596 \global\let\bbl@inidata\empty
3597 \bbl@exp{\bbl@add@list\bbl@ini@loaded{language}}%
3598 \bbl@tglobal\bbl@ini@loaded
3599 \fi}
3600 \def\bbl@read@ini@aux{%
3601     \let\bbl@savestrings\empty
3602     \let\bbl@savetoday\empty
3603     \let\bbl@savestate\empty
3604     \def\bbl@elt##1##2##3{%
3605         \def\bbl@section{##1}%
3606         \in@{=date.}{=##1}% Find a better place
3607         \ifin@
3608             \bbl@ini@calendar{##1}%
3609         \fi
3610         \bbl@ifunset{bbl@inikv@##1}{}%
3611         {\csname bbl@inikv@##1\endcsname{##2}{##3}}%
3612     \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.

```

3613 \def\bbl@extend@ini@aux#1{%
3614     \bbl@startcommands*{#1}{captions}%
3615     % Activate captions/... and modify exports
3616     \bbl@csarg\def{inikv@captions.licr}##1##2{%
3617         \setlocalecaption{#1}{##1}{##2}%
3618     \def\bbl@inikv@captions##1##2{%
3619         \bbl@ini@captions@aux{##1}{##2}%
3620     \def\bbl@stringdef##1##2{\gdef##1{##2}}%
3621     \def\bbl@exportkey##1##2##3{%
3622         \bbl@ifunset{bbl@kv@##2}{}%
3623         {\expandafter\ifx\csname bbl@kv@##2\endcsname\empty\else
3624             \bbl@exp{\global\let<bbl@##1@language>\<bbl@kv@##2>}%
3625         \fi}}%
3626     % As with \bbl@read@ini, but with some changes
3627     \bbl@read@ini@aux
3628     \bbl@ini@exports\tw@
3629     % Update inidata@lang by pretending the ini is read.
3630     \def\bbl@elt##1##2##3{%
3631         \def\bbl@section{##1}%

```

```

3632 \bbl@iniline##2=##3\bbl@iniline}%
3633 \csname bbl@inidata@#1\endcsname
3634 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
3635 \StartBabelCommands*{#1}{date}% And from the import stuff
3636 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
3637 \bbl@savetoday
3638 \bbl@savedate
3639 \bbl@endcommands}

```

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

```

3640 \def\bbl@ini@calendar#1{%
3641 \lowercase{\def\bbl@tempa{= #1 =}}%
3642 \bbl@replace\bbl@tempa{=date.gregorian}{}%
3643 \bbl@replace\bbl@tempa{=date.}{}%
3644 \in@{.licr=}{#1=}%
3645 \ifin@
3646 \ifcase\bbl@engine
3647 \bbl@replace\bbl@tempa{.licr=}{}%
3648 \else
3649 \let\bbl@tempa\relax
3650 \fi
3651 \fi
3652 \ifx\bbl@tempa\relax\else
3653 \bbl@replace\bbl@tempa{=}{}%
3654 \bbl@exp{%
3655 \def\<bbl@inikv@#1>####1####2{%
3656 \\\bbl@inidata####1...\relax{####2}{\bbl@tempa}}}%
3657 \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).

```

3658 \def\bbl@renewinikey#1/#2\@#3{%
3659 \edef\bbl@tempa{\zap@space #1 \@empty}% section
3660 \edef\bbl@tempb{\zap@space #2 \@empty}% key
3661 \bbl@trim\toks@{#3}% value
3662 \bbl@exp{%
3663 \edef\\bbl@key@list{\bbl@key@list \bbl@tempa\bbl@tempb;}%
3664 \\g@addto@macro\\bbl@inidata{%
3665 \\\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.

```

3666 \def\bbl@exportkey#1#2#3{%
3667 \bbl@ifunset{bbl@kv@#2}%
3668 {\bbl@csarg\gdef{#1@\languagename}{#3}}%
3669 {\expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
3670 \bbl@csarg\gdef{#1@\languagename}{#3}}%
3671 \else
3672 \bbl@exp{\global\let\<bbl@#1@\languagename>\<bbl@kv@#2>}%
3673 \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.

```

3674 \def\bbl@iniwarning#1{%
3675 \bbl@ifunset{bbl@kv@identification.warning#1}{}%
3676 {\bbl@warning{%

```

```

3677      From babel-\bbl@cs{lini@\languagename}.ini:\\%
3678      \bbl@cs{@kv@identification.warning#1}\\%
3679      Reported }}}
3680 %
3681 \let\bbl@release@transforms\@empty
3682 %
3683 \def\bbl@ini@exports#1{%
3684   % Identification always exported
3685   \bbl@iniwarning{}%
3686   \ifcase\bbl@engine
3687     \bbl@iniwarning{.pdflatex}%
3688   \or
3689     \bbl@iniwarning{.lualatex}%
3690   \or
3691     \bbl@iniwarning{.xelatex}%
3692   \fi%
3693   \bbl@exportkey{lllevel}{identification.load.level}{}%
3694   \bbl@exportkey{elname}{identification.name.english}{}%
3695   \bbl@exp{\\ \bbl@exportkey{lname}{identification.name.opentype}%
3696     {\csname bbl@elname@\languagename\endcsname}}%
3697   \bbl@exportkey{tbcpl}{identification.tag.bcp47}{}%
3698   \bbl@exportkey{lbcpl}{identification.language.tag.bcp47}{}%
3699   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3700   \bbl@exportkey{esname}{identification.script.name}{}%
3701   \bbl@exp{\\ \bbl@exportkey{sname}{identification.script.name.opentype}%
3702     {\csname bbl@esname@\languagename\endcsname}}%
3703   \bbl@exportkey{sbcpl}{identification.script.tag.bcp47}{}%
3704   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3705   % Also maps bcp47 -> languagename
3706   \ifbbl@bcptoname
3707     \bbl@csarg\xdef{bcp@map@\bbl@cl{tbcpl}}{\languagename}%
3708   \fi
3709   % Conditional
3710   \ifnum#1>\z@           % 0 = only info, 1, 2 = basic, (re)new
3711     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3712     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3713     \bbl@exportkey{lftth}{typography.lefthyphenmin}{2}%
3714     \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3715     \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3716     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3717     \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3718     \bbl@exportkey{intsp}{typography.intraspaces}{}%
3719     \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3720     \bbl@exportkey{chrng}{characters.ranges}{}%
3721     \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3722     \ifnum#1=\tw@       % only (re)new
3723       \bbl@exportkey{rqtex}{identification.require.babel}{}%
3724       \bbl@tglobal\bbl@savetoday
3725       \bbl@tglobal\bbl@savestate
3726       \bbl@savestrings
3727     \fi
3728   \fi}

```

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

```

3729 \def\bbl@inikv#1#2{%      key=value
3730   \toks@{#2}%             This hides #'s from ini values
3731   \bbl@csarg\xdef{@kv@\bbl@section.#1}{\the\toks@}}

```

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

```

3732 \let\bbl@inikv@identification\bbl@inikv
3733 \let\bbl@inikv@typography\bbl@inikv
3734 \let\bbl@inikv@characters\bbl@inikv
3735 \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’.

```

3736 \def\bbl@inikv@counters#1#2{%
3737   \bbl@ifsamestring{#1}{digits}%
3738   {\bbl@error{The counter name 'digits' is reserved for mapping\\%
3739             decimal digits}%
3740    {Use another name.}}%
3741   }%
3742   \def\bbl@tempc{#1}%
3743   \bbl@trim@def{\bbl@tempb*}{#2}%
3744   \in@{.1$}{#1$}%
3745   \ifin@
3746     \bbl@replace\bbl@tempc{.1}{}%
3747     \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}%
3748     \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3749   \fi
3750   \in@{.F.}{#1}%
3751   \ifin@else\in@{.S.}{#1}\fi
3752   \ifin@
3753     \bbl@csarg\protected@xdef{cntr@#1@\language}{\bbl@tempb*}%
3754   \else
3755     \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3756     \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3757     \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
3758   \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.

```

3759 \ifcase\bbl@engine
3760   \bbl@csarg\def{inikv@captions.licr}#1#2{%
3761     \bbl@ini@captions@aux{#1}{#2}}
3762 \else
3763   \def\bbl@inikv@captions#1#2{%
3764     \bbl@ini@captions@aux{#1}{#2}}
3765 \fi

```

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

```

3766 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3767   \bbl@replace\bbl@tempa{.template}{}%
3768   \def\bbl@toreplace{#1}{}%
3769   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
3770   \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3771   \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3772   \bbl@replace\bbl@toreplace{[ ]}{name\endcsname}}%
3773   \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
3774   \bbl@xin@{,\bbl@tempa,},{chapter,appendix,part,}%
3775   \ifin@
3776     \@nameuse{\bbl@patch\bbl@tempa}%
3777     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3778   \fi
3779   \bbl@xin@{,\bbl@tempa,},{figure,table,}%
3780   \ifin@

```

```

3781 \toks@\expandafter{\bbl@toreplace}%
3782 \bbl@exp{\gdef\<fnum@\bbl@tempa>\the\toks@}}%
3783 \fi}
3784 \def\bbl@ini@captions@aux#1#2{%
3785 \bbl@trim@def\bbl@tempa{#1}%
3786 \bbl@xin@{.template}{\bbl@tempa}%
3787 \ifin@
3788 \bbl@ini@captions@template{#2}\languagename
3789 \else
3790 \bbl@ifblank{#2}%
3791 {\bbl@exp{%
3792 \toks@{\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}%
3793 {\bbl@trim\toks@{#2}}}%
3794 \bbl@exp{%
3795 \bbl@add\bbl@savestrings{%
3796 \SetString\<\bbl@tempa name>\the\toks@}}%
3797 \toks@\expandafter{\bbl@captionslist}%
3798 \bbl@exp{\in@\<\bbl@tempa name>\the\toks@}}%
3799 \ifin@\else
3800 \bbl@exp{%
3801 \bbl@add\<\bbl@extracaps@\languagename>\<\bbl@tempa name>%
3802 \bbl@tglobal\<\bbl@extracaps@\languagename>%
3803 \fi
3804 \fi}

```

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

```

3805 \def\bbl@list@the{%
3806 part,chapter,section,subsection,subsubsection,paragraph,%
3807 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3808 table,page,footnote,mpfootnote,mpfn}
3809 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3810 \bbl@ifunset{bbl@map@#1@\languagename}%
3811 {\@nameuse{#1}}%
3812 {\@nameuse{bbl@map@#1@\languagename}}}
3813 \def\bbl@inikv@labels#1#2{%
3814 \in@{.map}{#1}%
3815 \ifin@
3816 \ifx\bbl@KVP@labels\@nil\else
3817 \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3818 \ifin@
3819 \def\bbl@tempc{#1}%
3820 \bbl@replace\bbl@tempc{.map}{}%
3821 \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3822 \bbl@exp{%
3823 \gdef\<\bbl@map@\bbl@tempc @\languagename>%
3824 {\ifin@\<#2>\else\\localecounter{#2}\fi}}%
3825 \bbl@foreach\bbl@list@the{%
3826 \bbl@ifunset{the##1}{}%
3827 {\bbl@exp{\let\bbl@tempd\<the##1>%
3828 \bbl@exp{%
3829 \bbl@sreplace\<the##1>%
3830 {\<\bbl@tempc>{##1}}{\bbl@map@cnt{\bbl@tempc}{##1}}}%
3831 \bbl@sreplace\<the##1>%
3832 {\<\@empty @\bbl@tempc>\<c##1>{\bbl@map@cnt{\bbl@tempc}{##1}}}%
3833 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3834 \toks@\expandafter\expandafter\expandafter{%
3835 \csname the##1\endcsname}%
3836 \expandafter\def\csname the##1\endcsname{\the\toks@}}%
3837 \fi}}%

```

```

3838     \fi
3839     \fi
3840 %
3841 \else
3842 %
3843 % The following code is still under study. You can test it and make
3844 % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3845 % language dependent.
3846 \in{enumerate.}{#1}%
3847 \ifin@
3848     \def\bbl@tempa{#1}%
3849     \bbl@replace\bbl@tempa{enumerate.}{}%
3850     \def\bbl@toreplace{#2}%
3851     \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3852     \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3853     \bbl@replace\bbl@toreplace{[ ]}{\endcsname{}}}%
3854     \toks@ \expandafter{\bbl@toreplace}%
3855     % TODO. Execute only once:
3856     \bbl@exp{%
3857         \\bbl@add\<extras\language>{%
3858             \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3859             \def\<labelenum\romannumeral\bbl@tempa>\the\toks@}%
3860         \\bbl@toglobal\<extras\language>}%
3861     \fi
3862 \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.

```

3863 \def\bbl@chapttype{chapter}
3864 \ifx\@makechapterhead\@undefined
3865     \let\bbl@patchchapter\relax
3866 \else\ifx\thechapter\@undefined
3867     \let\bbl@patchchapter\relax
3868 \else\ifx\ps@headings\@undefined
3869     \let\bbl@patchchapter\relax
3870 \else
3871     \def\bbl@patchchapter{%
3872         \global\let\bbl@patchchapter\relax
3873         \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3874         \bbl@toglobal\appendix
3875         \bbl@sreplace\ps@headings
3876             {\@chapapp\thechapter}%
3877             {\bbl@chapterformat}%
3878         \bbl@toglobal\ps@headings
3879         \bbl@sreplace\chaptermark
3880             {\@chapapp\thechapter}%
3881             {\bbl@chapterformat}%
3882         \bbl@toglobal\chaptermark
3883         \bbl@sreplace\@makechapterhead
3884             {\@chapapp\space\thechapter}%
3885             {\bbl@chapterformat}%
3886         \bbl@toglobal\@makechapterhead
3887         \gdef\bbl@chapterformat{%
3888             \bbl@ifunset{\bbl@bbl@chapttype fmt@\language}%
3889             {\@chapapp\space\thechapter}
3890             {\@nameuse{\bbl@bbl@chapttype fmt@\language}}}}
3891     \let\bbl@patchappendix\bbl@patchchapter

```

```

3892 \fi\fi\fi
3893 \ifx\@part\@undefined
3894   \let\bbl@patchpart\relax
3895 \else
3896   \def\bbl@patchpart{%
3897     \global\let\bbl@patchpart\relax
3898     \bbl@sreplace\@part
3899       {\partname\nobreakspace\thepart}%
3900       {\bbl@partformat}%
3901     \bbl@tglobal\@part
3902     \gdef\bbl@partformat{%
3903       \bbl@ifunset{\bbl@partfmt@\language\language}%
3904       {\partname\nobreakspace\thepart}
3905       {\@nameuse{\bbl@partfmt@\language\language}}}}
3906 \fi

```

#### **Date.** TODO. Document

```

3907 % Arguments are _not_ protected.
3908 \let\bbl@calendar\@empty
3909 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3910 \def\bbl@localedate#1#2#3#4{%
3911   \begin{group}
3912     \ifx\@empty#1\@empty\else
3913       \let\bbl@ld@calendar\@empty
3914       \let\bbl@ld@variant\@empty
3915       \edef\bbl@tempa{\zap@space#1 \@empty}%
3916       \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ld##1}{##2}}%
3917       \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
3918       \edef\bbl@calendar{%
3919         \bbl@ld@calendar
3920         \ifx\bbl@ld@variant\@empty\else
3921           .\bbl@ld@variant
3922         \fi}%
3923       \bbl@replace\bbl@calendar{\gregorian}{}%
3924     \fi
3925     \bbl@cased
3926     {\@nameuse{\bbl@date@\language\language @\bbl@calendar}{#2}{#3}{#4}}%
3927   \end{group}
3928 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3929 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3930   \bbl@trim@def\bbl@tempa{#1.#2}%
3931   \bbl@ifsamestring{\bbl@tempa}{months.wide}%      to savedate
3932   {\bbl@trim@def\bbl@tempa{#3}%
3933     \bbl@trim\toks@{#5}%
3934     \@temptokena\expandafter{\bbl@savedate}%
3935     \bbl@exp{% Reverse order - in ini last wins
3936       \def\\bbl@savedate{%
3937         \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3938         \the\@temptokena}}}%
3939   {\bbl@ifsamestring{\bbl@tempa}{date.long}%      defined now
3940     {\lowercase{\def\bbl@tempb{#6}}%
3941       \bbl@trim@def\bbl@toreplace{#5}%
3942       \bbl@TG@@date
3943       \bbl@ifunset{\bbl@date@\language\language @}%
3944       {\bbl@exp{% TODO. Move to a better place.
3945         \gdef\<\language\language date>{\protect\<\language\language date >}%
3946         \gdef\<\language\language date >####1####2####3{%
3947           \\bbl@usedategrouptrue
3948           \<\bbl@ensure@\language\language>{%

```



```

3949         \\localdate{####1}{###2}{###3}}}%
3950     \\bbl@add\\bbl@savetoday{%
3951         \\SetString\\today{%
3952             <\\language name date>%
3953             {\\the\\year}{\\the\\month}{\\the\\day}}}%
3954     }%
3955     \\global\\bbl@csarg\\let{date@\\language name @}\\bbl@toreplace
3956     \\ifx\\bbl@tempb\\empty\\else
3957         \\global\\bbl@csarg\\let{date@\\language name @}\\bbl@tempb}\\bbl@toreplace
3958     \\fi}%
3959     {}%

```

**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.

```

3960 \\let\\bbl@calendar\\empty
3961 \\newcommand\\BabelDateSpace{\\nobreakspace}
3962 \\newcommand\\BabelDateDot{.\\@} % TODO. \\let instead of repeating
3963 \\newcommand\\BabelDated[1]{\\number#1}
3964 \\newcommand\\BabelDatedd[1]{\\ifnum#1<10 0\\fi\\number#1}
3965 \\newcommand\\BabelDateM[1]{\\number#1}
3966 \\newcommand\\BabelDateMM[1]{\\ifnum#1<10 0\\fi\\number#1}
3967 \\newcommand\\BabelDateMMMM[1]{%
3968     \\csname month\\romannumeral#1\\bbl@calendar name\\endcsname}%
3969 \\newcommand\\BabelDatey[1]{\\number#1}%
3970 \\newcommand\\BabelDateyy[1]{%
3971     \\ifnum#1<10 0\\number#1 %
3972     \\else\\ifnum#1<100 \\number#1 %
3973     \\else\\ifnum#1<1000 \\expandafter\\@gobble\\number#1 %
3974     \\else\\ifnum#1<10000 \\expandafter\\@gobbletwo\\number#1 %
3975     \\else
3976         \\bbl@error
3977         {Currently two-digit years are restricted to the\\
3978         range 0-9999.}%
3979         {There is little you can do. Sorry.}%
3980     \\fi\\fi\\fi\\fi}
3981 \\newcommand\\BabelDateyyyy[1]{\\number#1} % FIXME - add leading 0
3982 \\def\\bbl@replace@finish@iii#1{%
3983     \\bbl@exp{\\def\\#1####1####2####3\\the\\toks@}}
3984 \\def\\bbl@TG@date{%
3985     \\bbl@replace\\bbl@toreplace{[ ]}{\\BabelDateSpace}}%
3986     \\bbl@replace\\bbl@toreplace{[.]}{\\BabelDateDot}}%
3987     \\bbl@replace\\bbl@toreplace{[d]}{\\BabelDated{####3}}%
3988     \\bbl@replace\\bbl@toreplace{[dd]}{\\BabelDatedd{####3}}%
3989     \\bbl@replace\\bbl@toreplace{[M]}{\\BabelDateM{####2}}%
3990     \\bbl@replace\\bbl@toreplace{[MM]}{\\BabelDateMM{####2}}%
3991     \\bbl@replace\\bbl@toreplace{[MMMM]}{\\BabelDateMMMM{####2}}%
3992     \\bbl@replace\\bbl@toreplace{[y]}{\\BabelDatey{####1}}%
3993     \\bbl@replace\\bbl@toreplace{[yy]}{\\BabelDateyy{####1}}%
3994     \\bbl@replace\\bbl@toreplace{[yyyy]}{\\BabelDateyyyy{####1}}%
3995     \\bbl@replace\\bbl@toreplace{[y]}{\\bbl@datecctr[####1|}%
3996     \\bbl@replace\\bbl@toreplace{[m]}{\\bbl@datecctr[####2|}%
3997     \\bbl@replace\\bbl@toreplace{[d]}{\\bbl@datecctr[####3|}%
3998 % Note after \\bbl@replace \\toks@ contains the resulting string.
3999 % TODO - Using this implicit behavior doesn't seem a good idea.
4000     \\bbl@replace@finish@iii\\bbl@toreplace}
4001 \\def\\bbl@datecctr{\\expandafter\\bbl@xdatecctr\\expandafter}
4002 \\def\\bbl@xdatecctr[#1|#2]{\\localnumeral{#2}{#1}}

```

**Transforms.**

```

4003 \let\bbl@release@transforms\@empty
4004 \@namedef{bbl@inikv@transforms.prehyphenation}{%
4005   \bbl@transforms\babelprehyphenation}
4006 \@namedef{bbl@inikv@transforms.posthyphenation}{%
4007   \bbl@transforms\babelposthyphenation}
4008 \def\bbl@transforms@aux#1#2#3,#4\relax{#1{#2}{#3}{#4}}
4009 \begingroup
4010   \catcode`\%=12
4011   \catcode`\&=14
4012   \gdef\bbl@transforms#1#2#3{&%
4013     \ifx\bbl@KVP@transforms\@nil\else
4014       \directlua{
4015         str = [==[#2]==]
4016         str = str:gsub('%.%d+%.%d+$', '')
4017         tex.print([[ \def\string\babeltempa{]] .. str .. [{}]])
4018       }&%
4019       \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
4020       \ifin@
4021         \in@{.0$}{#2$}&%
4022         \ifin@
4023           \g@addto@macro\bbl@release@transforms{&%
4024             \relax\bbl@transforms@aux#1{\language}\{#3}&%
4025           \else
4026             \g@addto@macro\bbl@release@transforms{, {#3}}&%
4027           \fi
4028         \fi
4029       \fi}
4030 \endgroup

```

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

```

4031 \def\bbl@provide@lsys#1{%
4032   \bbl@ifunset{bbl@lname@#1}%
4033     {\bbl@load@info{#1}}%
4034   }%
4035   \bbl@csarg\let{lsys@#1}\@empty
4036   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
4037   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
4038   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
4039   \bbl@ifunset{bbl@lname@#1}%
4040     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
4041   \ifcase\bbl@engine\or\or
4042     \bbl@ifunset{bbl@prehc@#1}{}%
4043     {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
4044     }%
4045     {\ifx\bbl@xenoxyph\@undefined
4046       \let\bbl@xenoxyph\bbl@xenoxyph@d
4047       \ifx\AtBeginDocument\@notprerr
4048         \expandafter\@secondoftwo % to execute right now
4049       \fi
4050       \AtBeginDocument{%
4051         \expandafter\bbl@add
4052         \csname selectfont \endcsname{\bbl@xenoxyph}%
4053         \expandafter\selectlanguage\expandafter{\language}%
4054         \expandafter\bbl@toggle\csname selectfont \endcsname}%
4055       \fi}}%
4056   \fi
4057   \bbl@csarg\bbl@toggle\lsys@#1}
4058 \def\bbl@xenoxyph@d{%

```

```

4059 \bbl@ifset{\bbl@prehc\language}%
4060 {\ifnum\hyphenchar\font=\defaultshyphenchar
4061   \iffontchar\font\bbl@cl{prehc}\relax
4062   \hyphenchar\font\bbl@cl{prehc}\relax
4063   \else\iffontchar\font"200B
4064     \hyphenchar\font"200B
4065   \else
4066     \bbl@warning
4067     {Neither 0 nor ZERO WIDTH SPACE are available\\%
4068      in the current font, and therefore the hyphen\\%
4069      will be printed. Try changing the fontspec's\\%
4070      'HyphenChar' to another value, but be aware\\%
4071      this setting is not safe (see the manual)}}%
4072   \hyphenchar\font\defaultshyphenchar
4073 \fi\fi
4074 \fi}%
4075 {\hyphenchar\font\defaultshyphenchar}}
4076 % \fi}

```

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

```

4077 \def\bbl@load@info#1{%
4078   \def\BabelBeforeIni##1##2{%
4079     \begingroup
4080       \bbl@read@ini{##1}0%
4081       \endinput           % babel- .tex may contain onlypreamble's
4082       \endgroup}%        boxed, to avoid extra spaces:
4083   {\bbl@input@texini{#1}}}

```

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

```

4084 \def\bbl@setdigits#1#2#3#4#5{%
4085   \bbl@exp{%
4086     \def\<\language name digits>####1{%          ie, \langdigits
4087       \<\bbl@digits@\language name>#####1\\\nil}%
4088       \let\<\bbl@cntr@digits@\language name>\<\language name digits>%
4089       \def\<\language name counter>####1{%        ie, \langcounter
4090         \\\expandafter\<\bbl@counter@\language name>%
4091         \\\csname c@#####1\endcsname}%
4092       \def\<\bbl@counter@\language name>####1{% ie, \bbl@counter@lang
4093         \\\expandafter\<\bbl@digits@\language name>%
4094         \\\number#####1\\\nil}}}%
4095   \def\bbl@tempa##1##2##3##4##5{%
4096     \bbl@exp{%      Wow, quite a lot of hashes! :-(
4097       \def\<\bbl@digits@\language name>#####1{%
4098         \\\ifx#####1\\\nil           % ie, \bbl@digits@lang
4099         \\\else
4100           \\\ifx0#####1#1%
4101           \\\else\\\ifx1#####1#2%
4102           \\\else\\\ifx2#####1#3%
4103           \\\else\\\ifx3#####1#4%
4104           \\\else\\\ifx4#####1#5%
4105           \\\else\\\ifx5#####1##1%
4106           \\\else\\\ifx6#####1##2%
4107           \\\else\\\ifx7#####1##3%
4108           \\\else\\\ifx8#####1##4%

```



```

4157             Perhaps it doesn't exist}%
4158             {See the manual for details.}}%
4159     {\bbl@cs{\csname bbl@info@#1\endcsname @\languagename}}}%
4160 % \namedef{bbl@info@name.locale}{lname}
4161 \@namedef{bbl@info@tag.ini}{lini}
4162 \@namedef{bbl@info@name.english}{elname}
4163 \@namedef{bbl@info@name.opentype}{lname}
4164 \@namedef{bbl@info@tag.bcp47}{tbcpl}
4165 \@namedef{bbl@info@language.tag.bcp47}{lbcp}
4166 \@namedef{bbl@info@tag.opentype}{lotf}
4167 \@namedef{bbl@info@script.name}{esname}
4168 \@namedef{bbl@info@script.name.opentype}{sname}
4169 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
4170 \@namedef{bbl@info@script.tag.opentype}{sotf}
4171 \let\bbl@ensureinfo\@gobble
4172 \newcommand\BabelEnsureInfo{%
4173   \ifx\InputIfFileExists\@undefined\else
4174     \def\bbl@ensureinfo##1{%
4175       \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}}%
4176   \fi
4177   \bbl@foreach\bbl@loaded{%
4178     \def\languagename{##1}%
4179     \bbl@ensureinfo{##1}}}%

```

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`.

```

4180 \newcommand\getlocaleproperty{%
4181   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
4182 \def\bbl@getproperty@s#1#2#3{%
4183   \let#1\relax
4184   \def\bbl@elt##1##2##3{%
4185     \bbl@ifsamestring{##1/##2}{##3}%
4186     {\providecommand#1{##3}%
4187     \def\bbl@elt####1####2####3{}}}%
4188   {}}%
4189   \bbl@cs{inidata@#2}}%
4190 \def\bbl@getproperty@x#1#2#3{%
4191   \bbl@getproperty@s{#1}{#2}{#3}%
4192   \ifx#1\relax
4193     \bbl@error
4194     {Unknown key for locale '#2':\%
4195     #3\%
4196     \string#1 will be set to \relax}%
4197     {Perhaps you misspelled it.}%
4198   \fi}
4199 \let\bbl@ini@loaded\@empty
4200 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

## 10 Adjusting the Babel behavior

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

```

4201 \newcommand\babeladjust[1]{% TODO. Error handling.
4202   \bbl@forkv{#1}{%
4203     \bbl@ifunset{bbl@ADJ@##1@##2}%
4204     {\bbl@cs{ADJ@##1}{##2}}%
4205     {\bbl@cs{ADJ@##1@##2}}}
4206 %

```

```

4207 \def\bbl@adjust@lua#1#2{%
4208   \ifvmode
4209     \ifnum\currentgrouplevel=\z@
4210       \directlua{ Babel.#2 }%
4211       \expandafter\expandafter\expandafter\@gobble
4212       \fi
4213   \fi
4214   {\bbl@error   % The error is gobbled if everything went ok.
4215     {Currently, #1 related features can be adjusted only\\%
4216       in the main vertical list.}%
4217     {Maybe things change in the future, but this is what it is.}}}
4218 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
4219   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
4220 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
4221   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
4222 \@namedef{bbl@ADJ@bidi.text@on}{%
4223   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
4224 \@namedef{bbl@ADJ@bidi.text@off}{%
4225   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
4226 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
4227   \bbl@adjust@lua{bidi}{digits_mapped=true}}
4228 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
4229   \bbl@adjust@lua{bidi}{digits_mapped=false}}
4230 %
4231 \@namedef{bbl@ADJ@linebreak.sea@on}{%
4232   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
4233 \@namedef{bbl@ADJ@linebreak.sea@off}{%
4234   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
4235 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
4236   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
4237 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
4238   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
4239 \@namedef{bbl@ADJ@justify.arabic@on}{%
4240   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
4241 \@namedef{bbl@ADJ@justify.arabic@off}{%
4242   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
4243 %
4244 \def\bbl@adjust@layout#1{%
4245   \ifvmode
4246     #1%
4247     \expandafter\@gobble
4248   \fi
4249   {\bbl@error   % The error is gobbled if everything went ok.
4250     {Currently, layout related features can be adjusted only\\%
4251       in vertical mode.}%
4252     {Maybe things change in the future, but this is what it is.}}}
4253 \@namedef{bbl@ADJ@layout.tabular@on}{%
4254   \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}}
4255 \@namedef{bbl@ADJ@layout.tabular@off}{%
4256   \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}}
4257 \@namedef{bbl@ADJ@layout.lists@on}{%
4258   \bbl@adjust@layout{\let\list\bbl@NL@list}}
4259 \@namedef{bbl@ADJ@layout.lists@off}{%
4260   \bbl@adjust@layout{\let\list\bbl@OL@list}}
4261 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
4262   \bbl@activateposthyphen}
4263 %
4264 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
4265   \bbl@bcpallowedtrue}

```

```

4266 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
4267   \bbl@bcpallowedfalse}
4268 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
4269   \def\bbl@bcp@prefix{#1}}
4270 \def\bbl@bcp@prefix{bcp47-}
4271 \@namedef{bbl@ADJ@autoload.options}#1{%
4272   \def\bbl@autoload@options{#1}}
4273 \let\bbl@autoload@bcptoptions\@empty
4274 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
4275   \def\bbl@autoload@bcptoptions{#1}}
4276 \newif\ifbbl@bcptoname
4277 \@namedef{bbl@ADJ@bcp47.toname@on}{%
4278   \bbl@bcptonametrue
4279   \BabelEnsureInfo}
4280 \@namedef{bbl@ADJ@bcp47.toname@off}{%
4281   \bbl@bcptonamefalse}
4282 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
4283   \directlua{ Babel.ignore_pre_char = function(node)
4284     return (node.lang == \the\csname l@nohyphenation\endcsname)
4285   end }}
4286 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
4287   \directlua{ Babel.ignore_pre_char = function(node)
4288     return false
4289   end }}
4290 % TODO: use babel name, override
4291 %
4292 % As the final task, load the code for lua.
4293 %
4294 \ifx\directlua\@undefined\else
4295   \ifx\bbl@luapatterns\@undefined
4296     \input luababel.def
4297   \fi
4298 \fi
4299 \</core>

A proxy file for switch.def

4300 \<*kernel>
4301 \let\bbl@onlyswitch\@empty
4302 \input babel.def
4303 \let\bbl@onlyswitch\@undefined
4304 \</kernel>
4305 \<*patterns>

```

## 11 Loading hyphenation patterns

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

To make sure that `LATEX 2.09` executes the `\@begindocumenthook` we would want to alter `\begin{document}`, but as this done too often already, we add the new code at the front of `\@preamblecmds`. But we can only do that after it has been defined, so we add this piece of code to `\dump`.

This new definition starts by adding an instruction to write a message on the terminal and in the transcript file to inform the user of the preloaded hyphenation patterns.

Then everything is restored to the old situation and the format is dumped.

```

4306 \<<Make sure ProvidesFile is defined>>
4307 \ProvidesFile{hyphen.cfg}[\<<date>>] \<<version>> Babel hyphens]
4308 \xdef\bbl@format{\jobname}

```

```

4309 \def\bbl@version{\<\version\>}
4310 \def\bbl@date{\<\date\>}
4311 \ifx\AtBeginDocument\@undefined
4312   \def\@empty{}
4313   \let\orig@dump\dump
4314   \def\dump{%
4315     \ifx\@ztryfc\@undefined
4316     \else
4317       \toks0=\expandafter{\@preamblecmds}%
4318       \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
4319       \def\@begindocumenthook{}}%
4320   \fi
4321   \let\dump\orig@dump\let\orig@dump\@undefined\dump}
4322 \fi
4323 <<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.

```

4324 \def\process@line#1#2 #3 #4 {%
4325   \ifx=#1%
4326     \process@synonym{#2}%
4327   \else
4328     \process@language{#1#2}{#3}{#4}%
4329   \fi
4330   \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.

```

4331 \toks@{}
4332 \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.

```

4333 \def\process@synonym#1{%
4334   \ifnum\last@language=\m@ne
4335     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4336   \else
4337     \expandafter\chardef\csname l@#1\endcsname\last@language
4338     \wlog{\string\l@#1=\string\language\the\last@language}%
4339     \expandafter\let\csname #1hyphenmins\endcsname\expandafter\endcsname
4340     \csname\language\endcsname hyphenmins\endcsname
4341     \let\bbl@elt\relax
4342     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4343   \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`. T<sub>E</sub>X does not keep track of these assignments. Therefore we try to detect such assignments and store them in the `\(lang)hyphenmins` macro. When no assignments were made we provide a default setting.

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

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

`\bbl@languages` saves a snapshot of the loaded languages in the form

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

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

```

4344 \def\process@language#1#2#3{%
4345   \expandafter\addlanguage\csname l@#1\endcsname
4346   \expandafter\language\csname l@#1\endcsname
4347   \edef\language{#1}%
4348   \bbl@hook@everylanguage{#1}%
4349   % > luatex
4350   \bbl@get@enc#1::\@@@
4351   \begingroup
4352     \lefthyphenmin\m@ne
4353     \bbl@hook@loadpatterns{#2}%
4354     % > luatex
4355     \ifnum\lefthyphenmin=\m@ne
4356     \else
4357       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4358         \the\lefthyphenmin\the\righthyphenmin}%
4359     \fi
4360   \endgroup
4361   \def\bbl@tempa{#3}%
4362   \ifx\bbl@tempa\@empty\else
4363     \bbl@hook@loadexceptions{#3}%
4364     % > luatex
4365   \fi
4366   \let\bbl@elt\relax
4367   \edef\bbl@languages{%
4368     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4369   \ifnum\the\language=\z@
4370     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4371       \set@hyphenmins\tw@\thr@@\relax
4372     \else
4373       \expandafter\expandafter\expandafter\set@hyphenmins
4374       \csname #1hyphenmins\endcsname
4375     \fi
4376     \the\toks@
4377     \toks@{}%
4378   \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.

```

4379 \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.

```

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

```

```

4435 \input luababel.def
4436 \fi
4437 \openin1 = babel-\bbl@format.cfg
4438 \ifeof1
4439 \else
4440 \input babel-\bbl@format.cfg\relax
4441 \fi
4442 \closein1
4443 \endgroup
4444 \bbl@hook@loadkernel{switch.def}

```

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

```

4445 \openin1 = language.dat

```

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

```

4446 \def\language{english}%
4447 \ifeof1
4448 \message{I couldn't find the file language.dat,\space
4449         I will try the file hyphen.tex}
4450 \input hyphen.tex\relax
4451 \chardef\l@english\z@
4452 \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`.

```

4453 \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.

```

4454 \loop
4455 \endlinechar\m@ne
4456 \read1 to \bbl@line
4457 \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.

```

4458 \if T\ifeof1F\fi T\relax
4459 \ifx\bbl@line\@empty\else
4460 \edef\bbl@line{\bbl@line\space\space\space}%
4461 \expandafter\process@line\bbl@line\relax
4462 \fi
4463 \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.

```

4464 \begingroup
4465 \def\bbl@elt#1#2#3#4{%
4466 \global\language=#2\relax
4467 \gdef\language{#1}%
4468 \def\bbl@elt##1##2##3##4{}}%
4469 \bbl@languages
4470 \endgroup
4471 \fi
4472 \closein1

```

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

```
4473 \if\the\toks@\else
4474 \errhelp{language.dat loads no language, only synonyms}
4475 \errmessage{Orphan language synonym}
4476 \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.

```
4477 \let\bbl@line\@undefined
4478 \let\process@line\@undefined
4479 \let\process@synonym\@undefined
4480 \let\process@language\@undefined
4481 \let\bbl@get@enc\@undefined
4482 \let\bbl@hyph@enc\@undefined
4483 \let\bbl@tempa\@undefined
4484 \let\bbl@hook@loadkernel\@undefined
4485 \let\bbl@hook@everylanguage\@undefined
4486 \let\bbl@hook@loadpatterns\@undefined
4487 \let\bbl@hook@loadexceptions\@undefined
4488 \</patterns>
```

Here the code for `iniTeX` ends.

## 12 Font handling with fontspec

Add the bidi handler just before `luaoftload`, which is loaded by default by LaTeX. Just in case, consider the possibility it has not been loaded. First, a couple of definitions related to bidi [misplaced].

```
4489 << *More package options >> ≡
4490 \chardef\bbl@bidimode\z@
4491 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4492 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4493 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4494 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4495 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4496 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4497 <</More package options >>
```

With explicit languages, we could define the font at once, but we don't. Just wait and see if the language is actually activated. `bbl@font` replaces hardcoded font names inside `\. . family` by the corresponding macro `\. . default`.

At the time of this writing, `fontspec` shows a warning about there are languages not available, which some people think refers to `babel`, even if there is nothing wrong. Here is hack to patch `fontspec` to avoid the misleading message, which is replaced by a more explanatory one.

```
4498 << *Font selection >> ≡
4499 \bbl@trace{Font handling with fontspec}
4500 \ifx\ExplSyntaxOn\@undefined\else
4501 \ExplSyntaxOn
4502 \catcode\ =10
4503 \def\bbl@loadfontspec{%
4504 \usepackage{fontspec}%
4505 \expandafter
4506 \def\csname msg~text~>~fontspec/language-not-exist\endcsname##1##2##3##4{%
4507 Font '\l_fontspec_fontname_tl' is using the\%
4508 default features for language '##1'.\%
4509 That's usually fine, because many languages\%
4510 require no specific features, but if the output is\%
4511 not as expected, consider selecting another font.}
```

```

4512 \expandafter
4513 \def\csname msg~text~>~fontspec/no-script\endcsname##1##2##3##4{%
4514   Font '\l_fontspec_fontname_tl' is using the\\%
4515   default features for script '##2'.\\%
4516   That's not always wrong, but if the output is\\%
4517   not as expected, consider selecting another font.}}
4518 \ExplSyntaxOff
4519 \fi
4520 \@onlypreamble\babelfont
4521 \newcommand\babelfont[2][{% 1=langs/scripts 2=fam
4522   \bbl@foreach{#1}{%
4523     \expandafter\ifx\csname date##1\endcsname\relax
4524       \IfFileExists{babel-##1.tex}%
4525       {\babelprovide{##1}}%
4526       {}%
4527     \fi}%
4528   \edef\bbl@tempa{#1}%
4529   \def\bbl@tempb{#2}% Used by \bbl@bblfont
4530   \ifx\fontspec\@undefined
4531     \bbl@loadfontspec
4532   \fi
4533   \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4534   \bbl@bblfont}
4535 \newcommand\bbl@bblfont[2][{% 1=features 2=fontname, @font=rm|sf|tt
4536   \bbl@ifunset{\bbl@tempb family}%
4537   {\bbl@providefam{\bbl@tempb}}%
4538   {\bbl@exp{%
4539     \\bbl@sreplace\<\bbl@tempb family >%
4540     {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
4541   % For the default font, just in case:
4542   \bbl@ifunset{\bbl@lsys\languagename}{\bbl@provide@lsys{\languagename}}}%
4543   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4544   {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
4545   \bbl@exp{%
4546     \let\<\bbl@tempb dflt@\languagename>\<\bbl@tempb dflt@>%
4547     \\bbl@font@set\<\bbl@tempb dflt@\languagename>%
4548     \<\bbl@tempb default>\<\bbl@tempb family>}}%
4549   {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4550     \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:

```

4551 \def\bbl@providefam#1{%
4552   \bbl@exp{%
4553     \\newcommand\<#1default>{}% Just define it
4554     \\bbl@add@list\\bbl@font@fams{#1}%
4555     \\DeclareRobustCommand\<#1family>{%
4556       \\not@math@alphabet\<#1family>\relax
4557       \\fontfamily\<#1default>\\selectfont}%
4558     \\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.

```

4559 \def\bbl@nostdfont#1{%
4560   \bbl@ifunset{\bbl@WFF@f@family}%
4561   {\bbl@csarg\gdef{\bbl@WFF@f@family}}}% Flag, to avoid dupl warns
4562   \bbl@infowarn{The current font is not a babel standard family:\\%
4563     #1%
4564     \fontname\font\\%
4565     There is nothing intrinsically wrong with this warning, and\\%

```

```

4566     you can ignore it altogether if you do not need these\\%
4567     families. But if they are used in the document, you should be\\%
4568     aware 'babel' will no set Script and Language for them, so\\%
4569     you may consider defining a new family with \string\babelfont.\\%
4570     See the manual for further details about \string\babelfont.\\%
4571     Reported}}
4572     {}}%
4573 \gdef\bbl@switchfont{%
4574   \bbl@ifunset\bbl@lsys@\language\name\{\bbl@provide@lsys\{\language\name\}\}%
4575   \bbl@exp{% eg Arabic -> arabic
4576     \lowercase{\edef\bbl@tempa{\bbl@cl{sname}}}}}%
4577   \bbl@foreach\bbl@font@fams{%
4578     \bbl@ifunset\bbl@##1dflt@\language\name\%      (1) language?
4579     {\bbl@ifunset\bbl@##1dflt@*\bbl@tempa\%      (2) from script?
4580       {\bbl@ifunset\bbl@##1dflt@\%      2=F - (3) from generic?
4581         {}%      123=F - nothing!
4582         {\bbl@exp{%      3=T - from generic
4583           \global\let\<\bbl@##1dflt@\language\name\>%
4584             \<\bbl@##1dflt@>}}}%
4585         {\bbl@exp{%      2=T - from script
4586           \global\let\<\bbl@##1dflt@\language\name\>%
4587             \<\bbl@##1dflt@*\bbl@tempa>}}}%
4588         {}%      1=T - language, already defined
4589     \def\bbl@tempa{\bbl@nostdfont{}}}%
4590   \bbl@foreach\bbl@font@fams{% don't gather with prev for
4591     \bbl@ifunset\bbl@##1dflt@\language\name\%
4592     {\bbl@cs{famrst@##1}%
4593       \global\bbl@csarg\let{famrst@##1}\relax}%
4594     {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4595       \bbl@add\originalTeX{%
4596         \bbl@font@rst{\bbl@cl{##1dflt}}}%
4597         \<##1default>\<##1family>{##1}}}%
4598       \bbl@font@set\<\bbl@##1dflt@\language\name\>% the main part!
4599       \<##1default>\<##1family>}}}%
4600   \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.

```

4601 \ifx\family\undefined\else % if latex
4602 \ifcase\bbl@engine % if pdftex
4603   \let\bbl@ckeckstdfonts\relax
4604 \else
4605   \def\bbl@ckeckstdfonts{%
4606     \begingroup
4607     \global\let\bbl@ckeckstdfonts\relax
4608     \let\bbl@tempa\@empty
4609     \bbl@foreach\bbl@font@fams{%
4610       \bbl@ifunset\bbl@##1dflt@{%
4611         {\nameuse{##1family}}%
4612         \bbl@csarg\gdef{WFF@\family}}}% Flag
4613       \bbl@exp{\bbl@add\bbbl@tempa{* \<##1family>= \family\\
4614         \space\space\fontname\font\\}}}%
4615       \bbl@csarg\xdef{##1dflt@}{\family}%
4616       \expandafter\xdef\csname ##1default\endcsname{\family}}}%
4617     {}}%
4618   \ifx\bbl@tempa\@empty\else
4619     \bbl@infowarn{The following font families will use the default\\
4620       settings for all or some languages:\\
4621       \bbl@tempa

```

```

4622         There is nothing intrinsically wrong with it, but\\%
4623         'babel' will no set Script and Language, which could\\%
4624         be relevant in some languages. If your document uses\\%
4625         these families, consider redefining them with \string\babelfont.\\%
4626         Reported}%
4627     \fi
4628 \endgroup}
4629 \fi
4630 \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.

```

4631 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4632 \bbl@xin@{<>}{#1}%
4633 \ifin@
4634 \bbl@exp{\bbl@fontspec@set\#1\expandafter@gobbletwo#1\#3}%
4635 \fi
4636 \bbl@exp{%          'Unprotected' macros return prev values
4637 \def\#2#1}%          eg, \rmdefault{\bbl@rmdflt@lang}
4638 \bbl@ifsamestring{#2}{f@family}%
4639 {\#3%
4640 \bbl@ifsamestring{f@series}{bfdefault}{\bfseries}}%
4641 \let\bbl@tempa\relax}%
4642 {}}
4643 %      TODO - next should be global?, but even local does its job. I'm
4644 %      still not sure -- must investigate:
4645 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4646 \let\bbl@tempe\bbl@mapselect
4647 \let\bbl@mapselect\relax
4648 \let\bbl@temp@fam#4%          eg, '\rmfamily', to be restored below
4649 \let#4\@empty %          Make sure \renewfontfamily is valid
4650 \bbl@exp{%
4651 \let\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
4652 <keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4653 {\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4654 <keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4655 {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4656 \renewfontfamily\#4%
4657 [\bbl@cs{lsys@\languagename},#2]{#3}% ie \bbl@exp{..}{#3}
4658 \begingroup
4659 #4%
4660 \xdef#1{f@family}%          eg, \bbl@rmdflt@lang{FreeSerif(0)}
4661 \endgroup
4662 \let#4\bbl@temp@fam
4663 \bbl@exp{\let<\bbl@stripslash#4\space>\bbl@temp@pfam
4664 \let\bbl@mapselect\bbl@tempe}%

```

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

```

4665 \def\bbl@font@rst#1#2#3#4{%
4666 \bbl@csarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}

```

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

```

4667 \def\bbl@font@fams{rm,sf,tt}

```

The old tentative way. Short and preverved for compatibility, but deprecated. Note there is no direct alternative for \babelFSfeatures. The reason in explained in the user guide, but essentially – that was not the way to go :-).

```

4668 \newcommand\babelFSstore[2][{%
4669   \bbl@ifblank{#1}%
4670   {\bbl@csarg\def{sname@#2}{Latin}}%
4671   {\bbl@csarg\def{sname@#2}{#1}}%
4672   \bbl@provide@dirs{#2}%
4673   \bbl@csarg\ifnum{wdir@#2}>\z@
4674     \let\bbl@beforeforeign\leavevmode
4675     \EnableBabelHook{babel-bidi}%
4676   \fi
4677   \bbl@foreach{#2}{%
4678     \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4679     \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4680     \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4681 \def\bbl@FSstore#1#2#3#4{%
4682   \bbl@csarg\edef{#2default#1}{#3}%
4683   \expandafter\addto\csname extras#1\endcsname{%
4684     \let#4#3%
4685     \ifx#3\f@family
4686       \edef#3{\csname bbl@#2default#1\endcsname}%
4687       \fontfamily{#3}\selectfont
4688     \else
4689       \edef#3{\csname bbl@#2default#1\endcsname}%
4690       \fi}%
4691   \expandafter\addto\csname noextras#1\endcsname{%
4692     \ifx#3\f@family
4693       \fontfamily{#4}\selectfont
4694       \fi
4695     \let#3#4}}
4696 \let\bbl@langfeatures\@empty
4697 \def\babelFSfeatures{% make sure \fontspec is redefined once
4698   \let\bbl@ori@fontspec\fontspec
4699   \renewcommand\fontspec[1][{%
4700     \bbl@ori@fontspec[\bbl@langfeatures##1]}
4701   \let\babelFSfeatures\bbl@FSfeatures
4702   \babelFSfeatures}
4703 \def\bbl@FSfeatures#1#2{%
4704   \expandafter\addto\csname extras#1\endcsname{%
4705     \babel@save\bbl@langfeatures
4706     \edef\bbl@langfeatures{#2,}}
4707 <</Font selection>>

```

## 13 Hooks for XeTeX and LuaTeX

### 13.1 XeTeX

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

```

4708 <<{*Footnote changes}>> ≡
4709 \bbl@trace{Bidi footnotes}
4710 \ifnum\bbl@bidimode>\z@
4711   \def\bbl@footnote#1#2#3{%
4712     \@ifnextchar[%
4713       {\bbl@footnote@o{#1}{#2}{#3}}%
4714       {\bbl@footnote@x{#1}{#2}{#3}}}
4715   \long\def\bbl@footnote@x#1#2#3#4{%
4716     \bgroup
4717     \select@language@x{\bbl@main@language}%
4718     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%

```



```

4719 \egroup}
4720 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4721 \bgroup
4722 \select@language@x{\bbl@main@language}%
4723 \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4724 \egroup}
4725 \def\bbl@footnotetext#1#2#3{%
4726 \@ifnextchar[%
4727 {\bbl@footnotetext@o{#1}{#2}{#3}}%
4728 {\bbl@footnotetext@x{#1}{#2}{#3}}}
4729 \long\def\bbl@footnotetext@x#1#2#3#4{%
4730 \bgroup
4731 \select@language@x{\bbl@main@language}%
4732 \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4733 \egroup}
4734 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4735 \bgroup
4736 \select@language@x{\bbl@main@language}%
4737 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4738 \egroup}
4739 \def\BabelFootnote#1#2#3#4{%
4740 \ifx\bbl@fn@footnote\undefined
4741 \let\bbl@fn@footnote\footnote
4742 \fi
4743 \ifx\bbl@fn@footnotetext\undefined
4744 \let\bbl@fn@footnotetext\footnotetext
4745 \fi
4746 \bbl@ifblank{#2}%
4747 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4748 \@namedef{\bbl@stripslash#1text}%
4749 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4750 {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4751 \@namedef{\bbl@stripslash#1text}%
4752 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4753 \fi
4754 <</Footnote changes>>

```

Now, the code.

```

4755 (*xetex)
4756 \def\BabelStringsDefault{unicode}
4757 \let\xebbl@stop\relax
4758 \AddBabelHook{xetex}{encodedcommands}{%
4759 \def\bbl@tempa{#1}%
4760 \ifx\bbl@tempa\empty
4761 \XeTeXinputencoding"bytes"%
4762 \else
4763 \XeTeXinputencoding"#1"%
4764 \fi
4765 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4766 \AddBabelHook{xetex}{stopcommands}{%
4767 \xebbl@stop
4768 \let\xebbl@stop\relax}
4769 \def\bbl@intraspace#1 #2 #3\@@{%
4770 \bbl@csarg\gdef{\xeisp@\languagename}%
4771 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4772 \def\bbl@intrapenalty#1\@@{%
4773 \bbl@csarg\gdef{\xeipn@\languagename}%
4774 {\XeTeXlinebreakpenalty #1\relax}}
4775 \def\bbl@provide@intraspace{%

```

```

4776 \bbl@xin@{/s}{/\bbl@cl{lbrk}}%
4777 \ifin@else\bbl@xin@{/c}{/\bbl@cl{lbrk}}\fi
4778 \ifin@
4779 \bbl@ifunset{\bbl@intsp@{language}}{%
4780   {\expandafter\ifx\csname bbl@intsp@{language}\endcsname\@empty\else
4781     \ifx\bbl@KVP@intraspace\@nil
4782       \bbl@exp{%
4783         \\bbl@intraspace\bbl@cl{intsp}\\\@}%
4784       \fi
4785       \ifx\bbl@KVP@intrapenalty\@nil
4786         \bbl@intrapenalty0\@@
4787       \fi
4788     \fi
4789     \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4790       \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4791     \fi
4792     \ifx\bbl@KVP@intrapenalty\@nil\else
4793       \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4794     \fi
4795     \bbl@exp{%
4796       % TODO. Execute only once (but redundant):
4797       \\bbl@add{<extras\language>{%
4798         \XeTeXlinebreaklocale "\bbl@cl{tbc}"%
4799         \<bbl@xeisp@{language}>%
4800         \<bbl@xeipn@{language}>%
4801         \\bbl@tglobal{<extras\language>%
4802         \\bbl@add{<noextras\language>{%
4803           \XeTeXlinebreaklocale "en"%
4804           \\bbl@tglobal{<noextras\language>}%
4805         \ifx\bbl@ispace\@undefined
4806           \gdef\bbl@ispace{\bbl@cl{xeisp}}%
4807         \ifx\AtBeginDocument\@notprerr
4808           \expandafter\@secondoftwo % to execute right now
4809         \fi
4810         \AtBeginDocument{%
4811           \expandafter\bbl@add
4812           \csname selectfont \endcsname{\bbl@ispace}%
4813           \expandafter\bbl@tglobal\csname selectfont \endcsname}%
4814         \fi}%
4815     \fi}
4816 \ifx\DisableBabelHook\@undefined\endinput\fi
4817 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4818 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4819 \DisableBabelHook{babel-fontspec}
4820 <<Font selection>>
4821 \input txtbabel.def
4822 </xetex>

```

## 13.2 Layout

*In progress.*

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>.

```
4823 <*\texet>
```

```

4824 \providecommand\bbl@provide@intraspace{}
4825 \bbl@trace{Redefinitions for bidi layout}
4826 \def\bbl@sspre@caption{%
4827   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4828 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4829 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4830 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4831 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4832   \def\@hangfrom#1{%
4833     \setbox\@tempboxa\hbox{#1}%
4834     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4835     \noindent\box\@tempboxa}
4836   \def\raggedright{%
4837     \let\@centercr
4838     \bbl@startskip\z@skip
4839     \@rightskip\@flushglue
4840     \bbl@endskip\@rightskip
4841     \parindent\z@
4842     \parfillskip\bbl@startskip}
4843   \def\raggedleft{%
4844     \let\@centercr
4845     \bbl@startskip\@flushglue
4846     \bbl@endskip\z@skip
4847     \parindent\z@
4848     \parfillskip\bbl@endskip}
4849 \fi
4850 \IfBabelLayout{lists}
4851   {\bbl@sreplace\list
4852     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4853     \def\bbl@listleftmargin{%
4854       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4855     \ifcase\bbl@engine
4856       \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4857       \def\p@enumiii{\p@enumii}\theenumii{}
4858     \fi
4859     \bbl@sreplace\@verbatim
4860       {\leftskip\@totalleftmargin}%
4861       {\bbl@startskip\textwidth
4862         \advance\bbl@startskip-\linewidth}%
4863     \bbl@sreplace\@verbatim
4864       {\rightskip\z@skip}%
4865       {\bbl@endskip\z@skip}}%
4866   {}
4867 \IfBabelLayout{contents}
4868   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4869     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4870   {}
4871 \IfBabelLayout{columns}
4872   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4873     \def\bbl@outputbox#1{%
4874       \hb@xt@\textwidth{%
4875         \hskip\columnwidth
4876         \hfil
4877         {\normalcolor\vrule \@width\columnseprule}%
4878         \hfil
4879         \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4880         \hskip-\textwidth
4881         \hb@xt@\columnwidth{\box\@outputbox \hss}%
4882         \hskip\columnsep

```

```

4883 \hskip\columnwidth}}}%
4884 {}
4885 <<Footnote changes>>
4886 \IfBabelLayout{footnotes}%
4887 {\BabelFootnote\footnote\language\language{}{}}%
4888 \BabelFootnote\localfootnote\language\language{}{}}%
4889 \BabelFootnote\mainfootnote{}{}}{}
4890 {}

```

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.

```

4891 \IfBabelLayout{counters}%
4892 {\let\bbl@latin@arabic=\@arabic
4893 \def\@arabic#1{\babelsublr{\bbl@latin@arabic#1}}}%
4894 \let\bbl@asci@roman=\@roman
4895 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asci@roman#1}}}%
4896 \let\bbl@asci@Roman=\@Roman
4897 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asci@Roman#1}}}}{}
4898 </texet>

```

### 13.3 LuaTeX

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

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

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

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

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

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

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

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

```

4899 <*\luatex>
4900 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4901 \bbl@trace{Read language.dat}
4902 \ifx\bbl@readstream\undefined

```

```

4903 \csname newread\endcsname\bbl@readstream
4904 \fi
4905 \beginngroup
4906 \toks@{}
4907 \count@ \z@ % 0=start, 1=0th, 2=normal
4908 \def\bbl@process@line#1#2 #3 #4 {%
4909   \ifx=#1%
4910     \bbl@process@synonym{#2}%
4911   \else
4912     \bbl@process@language{#1#2}{#3}{#4}%
4913   \fi
4914   \ignorespaces}
4915 \def\bbl@manylang{%
4916   \ifnum\bbl@last>\@ne
4917     \bbl@info{Non-standard hyphenation setup}%
4918   \fi
4919   \let\bbl@manylang\relax}
4920 \def\bbl@process@language#1#2#3{%
4921   \ifcase\count@
4922     \@ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
4923   \or
4924     \count@\tw@
4925   \fi
4926   \ifnum\count@=\tw@
4927     \expandafter\addlanguage\csname l@#1\endcsname
4928     \language\allocationnumber
4929     \chardef\bbl@last\allocationnumber
4930     \bbl@manylang
4931     \let\bbl@elt\relax
4932     \xdef\bbl@languages{%
4933       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4934   \fi
4935   \the\toks@
4936   \toks@{}}
4937 \def\bbl@process@synonym@aux#1#2{%
4938   \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4939   \let\bbl@elt\relax
4940   \xdef\bbl@languages{%
4941     \bbl@languages\bbl@elt{#1}{#2}{}}}%
4942 \def\bbl@process@synonym#1{%
4943   \ifcase\count@
4944     \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4945   \or
4946     \@ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4947   \else
4948     \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4949   \fi}
4950 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4951   \chardef\l@english\z@
4952   \chardef\l@USenglish\z@
4953   \chardef\bbl@last\z@
4954   \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
4955   \gdef\bbl@languages{%
4956     \bbl@elt{english}{0}{hyphen.tex}}%
4957   \bbl@elt{USenglish}{0}{}%
4958 \else
4959   \global\let\bbl@languages@format\bbl@languages
4960   \def\bbl@elt#1#2#3#4{% Remove all except language 0
4961     \ifnum#2>\z@\else

```

```

4962      \noexpand\bb1@elt{#1}{#2}{#3}{#4}%
4963      \fi}%
4964      \xdef\bb1@languages{\bb1@languages}%
4965      \fi
4966      \def\bb1@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4967      \bb1@languages
4968      \openin\bb1@readstream=language.dat
4969      \ifeof\bb1@readstream
4970      \bb1@warning{I couldn't find language.dat. No additional\\%
4971      patterns loaded. Reported}%
4972      \else
4973      \loop
4974      \endlinechar\m@ne
4975      \read\bb1@readstream to \bb1@line
4976      \endlinechar``^^M
4977      \if T\ifeof\bb1@readstream F\fi T\relax
4978      \ifx\bb1@line\@empty\else
4979      \edef\bb1@line{\bb1@line\space\space\space}%
4980      \expandafter\bb1@process@line\bb1@line\relax
4981      \fi
4982      \repeat
4983      \fi
4984 \endgroup
4985 \bb1@trace{Macros for reading patterns files}
4986 \def\bb1@get@enc#1:#2:#3\@@{\def\bb1@hyph@enc{#2}}
4987 \ifx\babelcatcodetablenum\@undefined
4988 \ifx\newcatcodetable\@undefined
4989 \def\babelcatcodetablenum{5211}
4990 \def\bb1@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4991 \else
4992 \newcatcodetable\babelcatcodetablenum
4993 \newcatcodetable\bb1@pattcodes
4994 \fi
4995 \else
4996 \def\bb1@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4997 \fi
4998 \def\bb1@luapatterns#1#2{%
4999 \bb1@get@enc#1::\@@@
5000 \setbox\z@\hbox\bgroup
5001 \begingroup
5002 \savecatcodetable\babelcatcodetablenum\relax
5003 \initcatcodetable\bb1@pattcodes\relax
5004 \catcodetable\bb1@pattcodes\relax
5005 \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
5006 \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
5007 \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
5008 \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
5009 \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
5010 \catcode`\`=12 \catcode`\'=12 \catcode`\`=12
5011 \input #1\relax
5012 \catcodetable\babelcatcodetablenum\relax
5013 \endgroup
5014 \def\bb1@tempa{#2}%
5015 \ifx\bb1@tempa\@empty\else
5016 \input #2\relax
5017 \fi
5018 \egroup}%
5019 \def\bb1@patterns@lua#1{%
5020 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax

```

```

5021 \csname l@#1\endcsname
5022 \edef\bbl@tempa{#1}%
5023 \else
5024 \csname l@#1:\f@encoding\endcsname
5025 \edef\bbl@tempa{#1:\f@encoding}%
5026 \fi\relax
5027 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5028 \@ifundefined{bbl@hyphendata@the\language}%
5029 {\def\bbl@elt##1##2##3##4{%
5030 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
5031 \def\bbl@tempb{##3}%
5032 \ifx\bbl@tempb\empty\else % if not a synonymous
5033 \def\bbl@tempc{##3}{##4}}%
5034 \fi
5035 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5036 \fi}%
5037 \bbl@languages
5038 \@ifundefined{bbl@hyphendata@the\language}%
5039 {\bbl@info{No hyphenation patterns were set for\%
5040 language '\bbl@tempa'. Reported}}%
5041 {\expandafter\expandafter\expandafter\bbl@luapatterns
5042 \csname bbl@hyphendata@the\language\endcsname}}}%
5043 \endinput\fi
5044 % Here ends \ifx\AddBabelHook\@undefined
5045 % A few lines are only read by hyphen.cfg
5046 \ifx\DisableBabelHook\@undefined
5047 \AddBabelHook{luatex}{everylanguage}{%
5048 \def\process@language##1##2##3{%
5049 \def\process@line####1####2 ####3 ####4 {}}}
5050 \AddBabelHook{luatex}{loadpatterns}{%
5051 \input #1\relax
5052 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5053 {{#1}}}%
5054 \AddBabelHook{luatex}{loadexceptions}{%
5055 \input #1\relax
5056 \def\bbl@tempb##1##2{{#1}{#1}}%
5057 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5058 {\expandafter\expandafter\expandafter\bbl@tempb
5059 \csname bbl@hyphendata@the\language\endcsname}}
5060 \endinput\fi
5061 % Here stops reading code for hyphen.cfg
5062 % The following is read the 2nd time it's loaded
5063 \begingroup % TODO - to a lua file
5064 \catcode`\%=12
5065 \catcode`\'=12
5066 \catcode`\%=12
5067 \catcode`\:=12
5068 \directlua{
5069 Babel = Babel or {}
5070 function Babel.bytes(line)
5071 return line:gsub(".",
5072 function (chr) return unicode.utf8.char(string.byte(chr)) end)
5073 end
5074 function Babel.begin_process_input()
5075 if luatexbase and luatexbase.add_to_callback then
5076 luatexbase.add_to_callback('process_input_buffer',
5077 Babel.bytes, 'Babel.bytes')
5078 else
5079 Babel.callback = callback.find('process_input_buffer')

```

```

5080     callback.register('process_input_buffer',Babel.bytes)
5081 end
5082 end
5083 function Babel.end_process_input ()
5084     if luatexbase and luatexbase.remove_from_callback then
5085         luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
5086     else
5087         callback.register('process_input_buffer',Babel.callback)
5088     end
5089 end
5090 function Babel.addpatterns(pp, lg)
5091     local lg = lang.new(lg)
5092     local pats = lang.patterns(lg) or ''
5093     lang.clear_patterns(lg)
5094     for p in pp:gmatch('[^%s]+') do
5095         ss = ''
5096         for i in string.utfcharacters(p:gsub('%d', '')) do
5097             ss = ss .. '%d?' .. i
5098         end
5099         ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
5100         ss = ss:gsub('%.%%d%?$', '%%.')
5101         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5102         if n == 0 then
5103             tex.sprint(
5104                 [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5105                 .. p .. [[{}]])
5106             pats = pats .. ' ' .. p
5107         else
5108             tex.sprint(
5109                 [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5110                 .. p .. [[{}]])
5111         end
5112     end
5113     lang.patterns(lg, pats)
5114 end
5115 }
5116 \endgroup
5117 \ifx\newattribute\undefined\else
5118     \newattribute\bbl@attr@locale
5119     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale'}
5120     \AddBabelHook{luatex}{beforeextras}{%
5121         \setattribute\bbl@attr@locale\localeid}
5122 \fi
5123 \def\BabelStringsDefault{unicode}
5124 \let\luabbl@stop\relax
5125 \AddBabelHook{luatex}{encodedcommands}{%
5126     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5127     \ifx\bbl@tempa\bbl@tempb\else
5128         \directlua{Babel.begin_process_input()}%
5129         \def\luabbl@stop{%
5130             \directlua{Babel.end_process_input()}}%
5131     \fi}%
5132 \AddBabelHook{luatex}{stopcommands}{%
5133     \luabbl@stop
5134     \let\luabbl@stop\relax}
5135 \AddBabelHook{luatex}{patterns}{%
5136     \@ifundefined{bbl@hyphendata@the\language}%
5137     {\def\bbl@elt##1##2##3##4{%
5138         \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...

```



```

5139     \def\bbl@tempb{##3}%
5140     \ifx\bbl@tempb\@empty\else % if not a synonymous
5141       \def\bbl@tempc{##3}{##4}}%
5142     \fi
5143     \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5144   \fi}%
5145 \bbl@languages
5146 \@ifundefined{bbl@hyphendata@the\language}%
5147   {\bbl@info{No hyphenation patterns were set for\%
5148     language '#2'. Reported}}%
5149   {\expandafter\expandafter\expandafter\bbl@luapatterns
5150     \csname bbl@hyphendata@the\language\endcsname}}}%
5151 \@ifundefined{bbl@patterns@}{}%
5152 \begingroup
5153   \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5154   \ifin@else
5155     \ifx\bbl@patterns@\@empty\else
5156       \directlua{ Babel.addpatterns(
5157         [[\bbl@patterns@]], \number\language) }%
5158     \fi
5159     \@ifundefined{bbl@patterns@#1}%
5160     \@empty
5161     {\directlua{ Babel.addpatterns(
5162       [[\space\csname bbl@patterns@#1\endcsname]],
5163       \number\language) }}%
5164     \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5165   \fi
5166 \endgroup}%
5167 \bbl@exp{%
5168   \bbl@ifunset{bbl@prehc@\language\name}{}%
5169   {\bbl@ifblank{\bbl@cs{prehc@\language\name}}{}}%
5170   {\prehyphenchar=\bbl@c1{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.

```

5171 \@onlypreamble\babelpatterns
5172 \AtEndOfPackage{%
5173   \newcommand\babelpatterns[2][\@empty]{%
5174     \ifx\bbl@patterns\relax
5175       \let\bbl@patterns@\@empty
5176     \fi
5177     \ifx\bbl@pttnlist\@empty\else
5178       \bbl@warning{%
5179         You must not intermingle \string\selectlanguage\space and\%
5180         \string\babelpatterns\space or some patterns will not\%
5181         be taken into account. Reported}%
5182       \fi
5183       \ifx\@empty#1%
5184         \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5185       \else
5186         \edef\bbl@tempb{\zap@space#1 \@empty}%
5187         \bbl@for\bbl@tempa\bbl@tempb{%
5188           \bbl@fixname\bbl@tempa
5189           \bbl@iflanguage\bbl@tempa{%
5190             \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5191               \@ifundefined{bbl@patterns@\bbl@tempa}%
5192               \@empty
5193               {\csname bbl@patterns@\bbl@tempa\endcsname\space}%

```

```

5194         #2}}}%
5195     \fi}}

```

## 13.4 Southeast Asian scripts

First, some general code for line breaking, used by `\babelposthyphenation`.

Replace regular (ie, implicit) discretionary by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionary are not touched. See Unicode UAX 14.

```

5196% TODO - to a lua file
5197\directlua{
5198    Babel = Babel or {}
5199    Babel.linebreaking = Babel.linebreaking or {}
5200    Babel.linebreaking.before = {}
5201    Babel.linebreaking.after = {}
5202    Babel.locale = {} % Free to use, indexed by \localeid
5203    function Babel.linebreaking.add_before(func)
5204        tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5205        table.insert(Babel.linebreaking.before, func)
5206    end
5207    function Babel.linebreaking.add_after(func)
5208        tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5209        table.insert(Babel.linebreaking.after, func)
5210    end
5211}
5212\def\bbl@intraspace#1 #2 #3\@@{%
5213    \directlua{
5214        Babel = Babel or {}
5215        Babel.intraspaces = Babel.intraspaces or {}
5216        Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5217            {b = #1, p = #2, m = #3}
5218        Babel.locale_props[\the\localeid].intraspace = %
5219            {b = #1, p = #2, m = #3}
5220    }}
5221\def\bbl@intrapenalty#1\@@{%
5222    \directlua{
5223        Babel = Babel or {}
5224        Babel.intrapenalties = Babel.intrapenalties or {}
5225        Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5226        Babel.locale_props[\the\localeid].intrapenalty = #1
5227    }}
5228\begingroup
5229\catcode`\%=12
5230\catcode`\^=14
5231\catcode`\'=12
5232\catcode`\~=12
5233\gdef\bbl@seaintraspace{^
5234    \let\bbl@seaintraspace\relax
5235    \directlua{
5236        Babel = Babel or {}
5237        Babel.sea_enabled = true
5238        Babel.sea_ranges = Babel.sea_ranges or {}
5239        function Babel.set_chranges (script, chrng)
5240            local c = 0
5241            for s, e in string.gmatch(chrng..' ', '(.-%.(-)%s') do
5242                Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5243                c = c + 1
5244            end

```

```

5245 end
5246 function Babel.sea_disc_to_space (head)
5247     local sea_ranges = Babel.sea_ranges
5248     local last_char = nil
5249     local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5250     for item in node.traverse(head) do
5251         local i = item.id
5252         if i == node.id'glyph' then
5253             last_char = item
5254         elseif i == 7 and item.subtype == 3 and last_char
5255             and last_char.char > 0x0C99 then
5256             quad = font.getfont(last_char.font).size
5257             for lg, rg in pairs(sea_ranges) do
5258                 if last_char.char > rg[1] and last_char.char < rg[2] then
5259                     lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyril1
5260                     local intraspace = Babel.intraspaces[lg]
5261                     local intrapenalty = Babel.intrapenalties[lg]
5262                     local n
5263                     if intrapenalty ~= 0 then
5264                         n = node.new(14, 0)      ^% penalty
5265                         n.penalty = intrapenalty
5266                         node.insert_before(head, item, n)
5267                     end
5268                     n = node.new(12, 13)      ^% (glue, spaceskip)
5269                     node.setglue(n, intraspace.b * quad,
5270                                 intraspace.p * quad,
5271                                 intraspace.m * quad)
5272                     node.insert_before(head, item, n)
5273                     node.remove(head, item)
5274                 end
5275             end
5276         end
5277     end
5278 end
5279 }^^
5280 \bbl@luahyphenate}

```

## 13.5 CJK line breaking

Minimal line breaking for CJK scripts, mainly intended for simple documents and short texts as a secondary language. Only line breaking, with a little stretching for justification, without any attempt to adjust the spacing. It is based on (but does not strictly follow) the Unicode algorithm.

We first need a little table with the corresponding line breaking properties. A few characters have an additional key for the width (fullwidth vs. halfwidth), not yet used. There is a separate file, defined below.

```

5281 \catcode`\%=14
5282 \gdef\bbl@cjkintraspace{%
5283     \let\bbl@cjkintraspace\relax
5284     \directlua{
5285         Babel = Babel or {}
5286         require('babel-data-cjk.lua')
5287         Babel.cjk_enabled = true
5288         function Babel.cjk_linebreak(head)
5289             local GLYPH = node.id'glyph'
5290             local last_char = nil
5291             local quad = 655360      % 10 pt = 655360 = 10 * 65536
5292             local last_class = nil
5293             local last_lang = nil

```

```

5294
5295     for item in node.traverse(head) do
5296         if item.id == GLYPH then
5297
5298             local lang = item.lang
5299
5300             local LOCALE = node.get_attribute(item,
5301                 luatexbase.registernumber'bbl@attr@locale')
5302             local props = Babel.locale_props[LOCALE]
5303
5304             local class = Babel.cjk_class[item.char].c
5305
5306             if class == 'cp' then class = 'cl' end % ]) as CL
5307             if class == 'id' then class = 'I' end
5308
5309             local br = 0
5310             if class and last_class and Babel.cjk_breaks[last_class][class] then
5311                 br = Babel.cjk_breaks[last_class][class]
5312             end
5313
5314             if br == 1 and props.linebreak == 'c' and
5315                 lang ~= \the\l@nohyphenation\space and
5316                 last_lang ~= \the\l@nohyphenation then
5317                 local intrapenalty = props.intrapenalty
5318                 if intrapenalty ~= 0 then
5319                     local n = node.new(14, 0)    % penalty
5320                     n.penalty = intrapenalty
5321                     node.insert_before(head, item, n)
5322                 end
5323                 local intraspace = props.intraspace
5324                 local n = node.new(12, 13)    % (glue, spaceskip)
5325                 node.setglue(n, intraspace.b * quad,
5326                     intraspace.p * quad,
5327                     intraspace.m * quad)
5328                 node.insert_before(head, item, n)
5329             end
5330
5331             if font.getfont(item.font) then
5332                 quad = font.getfont(item.font).size
5333             end
5334             last_class = class
5335             last_lang = lang
5336             else % if penalty, glue or anything else
5337                 last_class = nil
5338             end
5339         end
5340         lang.hyphenate(head)
5341     end
5342 }%
5343 \bbl@luahyphenate}
5344 \gdef\bbl@luahyphenate{%
5345     \let\bbl@luahyphenate\relax
5346     \directlua{
5347         luatexbase.add_to_callback('hyphenate',
5348             function (head, tail)
5349                 if Babel.linebreaking.before then
5350                     for k, func in ipairs(Babel.linebreaking.before) do
5351                         func(head)
5352                     end

```

```

5353     end
5354     if Babel.cjk_enabled then
5355         Babel.cjk_linebreak(head)
5356     end
5357     lang.hyphenate(head)
5358     if Babel.linebreaking.after then
5359         for k, func in ipairs(Babel.linebreaking.after) do
5360             func(head)
5361         end
5362     end
5363     if Babel.sea_enabled then
5364         Babel.sea_disc_to_space(head)
5365     end
5366 end,
5367 'Babel.hyphenate')
5368 }
5369 }
5370 \endgroup
5371 \def\bbl@provide@intraspace{%
5372 \bbl@ifunset\bbl@intsp@{language}{}%
5373 {\expandafter\ifx\csname\bbl@intsp@{language}\endcsname\@empty\else
5374 \bbl@xin@{/c}{/\bbl@cl{lncr}}}%
5375 \ifin@ % cjk
5376 \bbl@cjk@intraspace
5377 \directlua{
5378     Babel = Babel or {}
5379     Babel.locale_props = Babel.locale_props or {}
5380     Babel.locale_props[\the\localeid].linebreak = 'c'
5381 }%
5382 \bbl@exp{\\bbl@intraspace\bbl@cl{intsp}\\@}%
5383 \ifx\bbl@KVP@intrapenalty\@nil
5384 \bbl@intrapenalty0\@
5385 \fi
5386 \else % sea
5387 \bbl@sea@intraspace
5388 \bbl@exp{\\bbl@intraspace\bbl@cl{intsp}\\@}%
5389 \directlua{
5390     Babel = Babel or {}
5391     Babel.sea_ranges = Babel.sea_ranges or {}
5392     Babel.set_chranges('\bbl@cl{sbc}',
5393                       '\bbl@cl{chrng}')
5394 }%
5395 \ifx\bbl@KVP@intrapenalty\@nil
5396 \bbl@intrapenalty0\@
5397 \fi
5398 \fi
5399 \fi
5400 \ifx\bbl@KVP@intrapenalty\@nil\else
5401 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
5402 \fi}}

```

## 13.6 Arabic justification

```

5403 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5404 \def\bblar@chars{%
5405 0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5406 0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5407 0640,0641,0642,0643,0644,0645,0646,0647,0649}
5408 \def\bblar@elongated{%

```

```

5409 0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5410 063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5411 0649,064A}
5412 \begingroup
5413 \catcode\_ =11 \catcode\`:=11
5414 \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5415 \endgroup
5416 \gdef\bbl@arabicjust{%
5417 \let\bbl@arabicjust\relax
5418 \newattribute\bblar@kashida
5419 \bblar@kashida=\z@
5420 \expandafter\bbl@add\csname selectfont \endcsname{{\bbl@parsejalt}}%
5421 \directlua{
5422 Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5423 Babel.arabic.elong_map[\the\localeid] = {}
5424 luatexbase.add_to_callback('post_linebreak_filter',
5425 Babel.arabic.justify, 'Babel.arabic.justify')
5426 luatexbase.add_to_callback('hpack_filter',
5427 Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5428 }}%
5429 % Save both node lists to make replacement. TODO. Save also widths to
5430 % make computations
5431 \def\bblar@fetchjalt#1#2#3#4{%
5432 \bbl@exp{\bbl@foreach{#1}}{%
5433 \bbl@ifunset\bblar@JE##1}%
5434 {\setbox\z@\hbox{^^^200d\char"##1#2}}%
5435 {\setbox\z@\hbox{^^^200d\char"@nameuse\bblar@JE##1#2}}%
5436 \directlua{%
5437 local last = nil
5438 for item in node.traverse(tex.box[0].head) do
5439 if item.id == node.id'glyph' and item.char > 0x600 and
5440 not (item.char == 0x200D) then
5441 last = item
5442 end
5443 end
5444 Babel.arabic.#3['##1#4'] = last.char
5445 }}}
5446 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5447 perhaps other tables (falt?, csw?). What about kaf? And diacritic
5448 % positioning?
5449 \gdef\bbl@parsejalt{%
5450 \ifx\addfontfeature\undefined\else
5451 \bbl@xin@{/e}{\bbl@c1{lnbrk}}%
5452 \ifin@
5453 \directlua{%
5454 if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5455 Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5456 tex.print([[string\csname\space bbl@parsejalti\endcsname]])
5457 end
5458 }%
5459 \fi
5460 \fi}
5461 \gdef\bbl@parsejalti{%
5462 \begingroup
5463 \let\bbl@parsejalt\relax % To avoid infinite loop
5464 \edef\bbl@tempb{\fontid\font}%
5465 \bblar@nofswarn
5466 \bblar@fetchjalt\bblar@elongated{{from}}{%
5467 \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura

```

```

5468 \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5469 \addfontfeature{RawFeature+=jalt}%
5470 % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5471 \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5472 \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5473 \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5474 \directlua{%
5475     for k, v in pairs(Babel.arabic.from) do
5476         if Babel.arabic.dest[k] and
5477             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5478             Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5479                 [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5480         end
5481     end
5482 }%
5483 \endgroup}
5484 %
5485 \begingroup
5486 \catcode`#=11
5487 \catcode`~=11
5488 \directlua{
5489
5490 Babel.arabic = Babel.arabic or {}
5491 Babel.arabic.from = {}
5492 Babel.arabic.dest = {}
5493 Babel.arabic.justify_factor = 0.95
5494 Babel.arabic.justify_enabled = true
5495
5496 function Babel.arabic.justify(head)
5497     if not Babel.arabic.justify_enabled then return head end
5498     for line in node.traverse_id(node.id'hlist', head) do
5499         Babel.arabic.justify_hlist(head, line)
5500     end
5501     return head
5502 end
5503
5504 function Babel.arabic.justify_hbox(head, gc, size, pack)
5505     local has_inf = false
5506     if Babel.arabic.justify_enabled and pack == 'exactly' then
5507         for n in node.traverse_id(12, head) do
5508             if n.stretch_order > 0 then has_inf = true end
5509         end
5510         if not has_inf then
5511             Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5512         end
5513     end
5514     return head
5515 end
5516
5517 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5518     local d, new
5519     local k_list, k_item, pos_inline
5520     local width, width_new, full, k_curr, wt_pos, goal, shift
5521     local subst_done = false
5522     local elong_map = Babel.arabic.elong_map
5523     local last_line
5524     local GLYPH = node.id'glyph'
5525     local KASHIDA = luatexbase.registernumber'bblar@kashida'
5526     local LOCALE = luatexbase.registernumber'bbl@attr@locale'

```

```

5527
5528 if line == nil then
5529     line = {}
5530     line.glue_sign = 1
5531     line.glue_order = 0
5532     line.head = head
5533     line.shift = 0
5534     line.width = size
5535 end
5536
5537 % Exclude last line. todo. But-- it discards one-word lines, too!
5538 % ? Look for glue = 12:15
5539 if (line.glue_sign == 1 and line.glue_order == 0) then
5540     elongs = {} % Stores elongated candidates of each line
5541     k_list = {} % And all letters with kashida
5542     pos_inline = 0 % Not yet used
5543
5544     for n in node.traverse_id(GLYPH, line.head) do
5545         pos_inline = pos_inline + 1 % To find where it is. Not used.
5546
5547         % Elongated glyphs
5548         if elong_map then
5549             local locale = node.get_attribute(n, LOCALE)
5550             if elong_map[locale] and elong_map[locale][n.font] and
5551                 elong_map[locale][n.font][n.char] then
5552                 table.insert(elongs, {node = n, locale = locale} )
5553                 node.set_attribute(n.prev, KASHIDA, 0)
5554             end
5555         end
5556
5557         % Tatwil
5558         if Babel.kashida_wts then
5559             local k_wt = node.get_attribute(n, KASHIDA)
5560             if k_wt > 0 then % todo. parameter for multi inserts
5561                 table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5562             end
5563         end
5564
5565     end % of node.traverse_id
5566
5567     if #elongs == 0 and #k_list == 0 then goto next_line end
5568     full = line.width
5569     shift = line.shift
5570     goal = full * Babel.arabic.justify_factor % A bit crude
5571     width = node.dimensions(line.head) % The 'natural' width
5572
5573     % == Elongated ==
5574     % Original idea taken from 'chickenize'
5575     while (#elongs > 0 and width < goal) do
5576         subst_done = true
5577         local x = #elongs
5578         local curr = elongs[x].node
5579         local oldchar = curr.char
5580         curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5581         width = node.dimensions(line.head) % Check if the line is too wide
5582         % Substitute back if the line would be too wide and break:
5583         if width > goal then
5584             curr.char = oldchar
5585             break

```



```

5586     end
5587     % If continue, pop the just substituted node from the list:
5588     table.remove(elongs, x)
5589 end
5590
5591 % == Tatwil ==
5592 if #k_list == 0 then goto next_line end
5593
5594 width = node.dimensions(line.head)    % The 'natural' width
5595 k_curr = #k_list
5596 wt_pos = 1
5597
5598 while width < goal do
5599     subst_done = true
5600     k_item = k_list[k_curr].node
5601     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5602         d = node.copy(k_item)
5603         d.char = 0x0640
5604         line.head, new = node.insert_after(line.head, k_item, d)
5605         width_new = node.dimensions(line.head)
5606         if width > goal or width == width_new then
5607             node.remove(line.head, new) % Better compute before
5608             break
5609         end
5610         width = width_new
5611     end
5612     if k_curr == 1 then
5613         k_curr = #k_list
5614         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5615     else
5616         k_curr = k_curr - 1
5617     end
5618 end
5619
5620 ::next_line::
5621
5622 % Must take into account marks and ins, see luatex manual.
5623 % Have to be executed only if there are changes. Investigate
5624 % what's going on exactly.
5625 if subst_done and not gc then
5626     d = node.hpack(line.head, full, 'exactly')
5627     d.shift = shift
5628     node.insert_before(head, line, d)
5629     node.remove(head, line)
5630 end
5631 end % if process line
5632 end
5633 }
5634 \endgroup
5635 \fi\fi % Arabic just block

```

### 13.7 Common stuff

```

5636 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5637 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfonts}
5638 \DisableBabelHook{babel-fontspec}
5639 <<Font selection>>

```

## 13.8 Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a short function which just traverse the node list to carry out the replacements. The table `loc_to_scr` gets the locale from a script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the `\language` and the `\localeid` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with `/` maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```
5640% TODO - to a lua file
5641\directlua{
5642Babel.script_blocks = {
5643  ['dflt'] = {},
5644  ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5645             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5646  ['Armn'] = {{0x0530, 0x058F}},
5647  ['Beng'] = {{0x0980, 0x09FF}},
5648  ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5649  ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5650  ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5651             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5652  ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5653  ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5654             {0xAB00, 0xAB2F}},
5655  ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5656  % Don't follow strictly Unicode, which places some Coptic letters in
5657  % the 'Greek and Coptic' block
5658  ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5659  ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5660             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5661             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5662             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5663             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5664             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5665  ['Hebr'] = {{0x0590, 0x05FF}},
5666  ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5667             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5668  ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5669  ['Knda'] = {{0x0C80, 0x0CFF}},
5670  ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5671             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5672             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5673  ['Lao0'] = {{0x0E80, 0x0EFF}},
5674  ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5675             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5676             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5677  ['Mahj'] = {{0x11150, 0x1117F}},
5678  ['Mlym'] = {{0x0D00, 0x0D7F}},
5679  ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5680  ['Orya'] = {{0x0B00, 0x0B7F}},
5681  ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5682  ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5683  ['Taml'] = {{0x0B80, 0x0BFF}},
5684  ['Telu'] = {{0x0C00, 0x0C7F}},
5685  ['Tfng'] = {{0x2D30, 0x2D7F}},
5686  ['Thai'] = {{0x0E00, 0x0E7F}},
5687  ['Tibt'] = {{0x0F00, 0x0FFF}},
5688  ['Vaii'] = {{0xA500, 0xA63F}},
5689  ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
```

```

5690 }
5691
5692 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5693 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5694 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5695
5696 function Babel.locale_map(head)
5697   if not Babel.locale_mapped then return head end
5698
5699   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
5700   local GLYPH = node.id('glyph')
5701   local inmath = false
5702   local toloc_save
5703   for item in node.traverse(head) do
5704     local toloc
5705     if not inmath and item.id == GLYPH then
5706       % Optimization: build a table with the chars found
5707       if Babel.chr_to_loc[item.char] then
5708         toloc = Babel.chr_to_loc[item.char]
5709       else
5710         for lc, maps in pairs(Babel.loc_to_scr) do
5711           for _, rg in pairs(maps) do
5712             if item.char >= rg[1] and item.char <= rg[2] then
5713               Babel.chr_to_loc[item.char] = lc
5714               toloc = lc
5715               break
5716             end
5717           end
5718         end
5719       end
5720       % Now, take action, but treat composite chars in a different
5721       % fashion, because they 'inherit' the previous locale. Not yet
5722       % optimized.
5723       if not toloc and
5724         (item.char >= 0x0300 and item.char <= 0x036F) or
5725         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5726         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5727         toloc = toloc_save
5728       end
5729       if toloc and toloc > -1 then
5730         if Babel.locale_props[toloc].lg then
5731           item.lang = Babel.locale_props[toloc].lg
5732           node.set_attribute(item, LOCALE, toloc)
5733         end
5734         if Babel.locale_props[toloc]['/'..item.font] then
5735           item.font = Babel.locale_props[toloc]['/'..item.font]
5736         end
5737         toloc_save = toloc
5738       end
5739     elseif not inmath and item.id == 7 then
5740       item.replace = item.replace and Babel.locale_map(item.replace)
5741       item.pre = item.pre and Babel.locale_map(item.pre)
5742       item.post = item.post and Babel.locale_map(item.post)
5743     elseif item.id == node.id'math' then
5744       inmath = (item.subtype == 0)
5745     end
5746   end
5747   return head
5748 end

```

5749 }

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

```

5750 \newcommand\babelcharproperty[1]{%
5751   \count@=#1\relax
5752   \ifvmode
5753     \expandafter\bb1@chprop
5754   \else
5755     \bb1@error{\string\babelcharproperty\space can be used only in\\%
5756               vertical mode (preamble or between paragraphs)}%
5757     {See the manual for futher info}%
5758   \fi}
5759 \newcommand\bb1@chprop[3][\the\count@]{%
5760   \@tempcnta=#1\relax
5761   \bb1@ifunset{\bb1@chprop@#2}%
5762   {\bb1@error{No property named '#2'. Allowed values are\\%
5763             direction (bc), mirror (bmg), and linebreak (lb)}%
5764   {See the manual for futher info}}%
5765   {}%
5766   \loop
5767     \bb1@cs{chprop@#2}{#3}%
5768     \ifnum\count@<\@tempcnta
5769       \advance\count@\@ne
5770     \repeat}
5771 \def\bb1@chprop@direction#1{%
5772   \directlua{
5773     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5774     Babel.characters[\the\count@]['d'] = '#1'
5775   }}
5776 \let\bb1@chprop@bc\bb1@chprop@direction
5777 \def\bb1@chprop@mirror#1{%
5778   \directlua{
5779     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5780     Babel.characters[\the\count@]['m'] = '\number#1'
5781   }}
5782 \let\bb1@chprop@bmg\bb1@chprop@mirror
5783 \def\bb1@chprop@linebreak#1{%
5784   \directlua{
5785     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5786     Babel.cjk_characters[\the\count@]['c'] = '#1'
5787   }}
5788 \let\bb1@chprop@lb\bb1@chprop@linebreak
5789 \def\bb1@chprop@locale#1{%
5790   \directlua{
5791     Babel.chr_to_loc = Babel.chr_to_loc or {}
5792     Babel.chr_to_loc[\the\count@] =
5793       \bb1@ifblank{#1}{-1000}{\the\bb1@cs{id@#1}}\space
5794   }}

```

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).

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: `str_to_nodes` converts the string returned by 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.

```

5795 \begingroup % TODO - to a lua file
5796 \catcode`\~ = 12
5797 \catcode`\# = 12
5798 \catcode`\% = 12
5799 \catcode`\& = 14
5800 \directlua{
5801   Babel.linebreaking.replacements = {}
5802   Babel.linebreaking.replacements[0] = {}  %% pre
5803   Babel.linebreaking.replacements[1] = {}  %% post
5804
5805   %% Discretionaries contain strings as nodes
5806   function Babel.str_to_nodes(fn, matches, base)
5807     local n, head, last
5808     if fn == nil then return nil end
5809     for s in string.utfvalues(fn(matches)) do
5810       if base.id == 7 then
5811         base = base.replace
5812       end
5813       n = node.copy(base)
5814       n.char = s
5815       if not head then
5816         head = n
5817       else
5818         last.next = n
5819       end
5820       last = n
5821     end
5822     return head
5823   end
5824
5825   Babel.fetch_subtext = {}
5826
5827   Babel.ignore_pre_char = function(node)
5828     return (node.lang == \the\l@nohyphenation)
5829   end
5830
5831   %% Merging both functions doesn't seem feasible, because there are too
5832   %% many differences.
5833   Babel.fetch_subtext[0] = function(head)
5834     local word_string = ''
5835     local word_nodes = {}
5836     local lang
5837     local item = head
5838     local inmath = false
5839
5840     while item do
5841
5842       if item.id == 11 then
5843         inmath = (item.subtype == 0)
5844       end
5845
5846       if inmath then
5847         %% pass
5848       elseif item.id == 29 then

```

```

5850         local locale = node.get_attribute(item, Babel.attr_locale)
5851
5852         if lang == locale or lang == nil then
5853             lang = lang or locale
5854             if Babel.ignore_pre_char(item) then
5855                 word_string = word_string .. Babel.us_char
5856             else
5857                 word_string = word_string .. unicode.utf8.char(item.char)
5858             end
5859             word_nodes[#word_nodes+1] = item
5860         else
5861             break
5862         end
5863
5864         elseif item.id == 12 and item.subtype == 13 then
5865             word_string = word_string .. ' '
5866             word_nodes[#word_nodes+1] = item
5867
5868             %% Ignore leading unrecognized nodes, too.
5869             elseif word_string ~= '' then
5870                 word_string = word_string .. Babel.us_char
5871                 word_nodes[#word_nodes+1] = item %% Will be ignored
5872             end
5873
5874             item = item.next
5875         end
5876
5877         %% Here and above we remove some trailing chars but not the
5878         %% corresponding nodes. But they aren't accessed.
5879         if word_string:sub(-1) == ' ' then
5880             word_string = word_string:sub(1,-2)
5881         end
5882         word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
5883         return word_string, word_nodes, item, lang
5884     end
5885
5886     Babel.fetch_subtext[1] = function(head)
5887         local word_string = ''
5888         local word_nodes = {}
5889         local lang
5890         local item = head
5891         local inmath = false
5892
5893         while item do
5894
5895             if item.id == 11 then
5896                 inmath = (item.subtype == 0)
5897             end
5898
5899             if inmath then
5900                 %% pass
5901             end
5902
5903             elseif item.id == 29 then
5904                 if item.lang == lang or lang == nil then
5905                     if (item.char ~= 124) and (item.char ~= 61) then %% not =, not |
5906                         lang = lang or item.lang
5907                         word_string = word_string .. unicode.utf8.char(item.char)
5908                         word_nodes[#word_nodes+1] = item
5909                     end

```

```

5909         else
5910             break
5911         end
5912
5913         elseif item.id == 7 and item.subtype == 2 then
5914             word_string = word_string .. '='
5915             word_nodes[#word_nodes+1] = item
5916
5917         elseif item.id == 7 and item.subtype == 3 then
5918             word_string = word_string .. '|'
5919             word_nodes[#word_nodes+1] = item
5920
5921         %% (1) Go to next word if nothing was found, and (2) implicitly
5922         %% remove leading USs.
5923         elseif word_string == '' then
5924             %% pass
5925
5926         %% This is the responsible for splitting by words.
5927         elseif (item.id == 12 and item.subtype == 13) then
5928             break
5929
5930         else
5931             word_string = word_string .. Babel.us_char
5932             word_nodes[#word_nodes+1] = item %% Will be ignored
5933         end
5934
5935         item = item.next
5936     end
5937
5938     word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
5939     return word_string, word_nodes, item, lang
5940 end
5941
5942 function Babel.pre_hyphenate_replace(head)
5943     Babel.hyphenate_replace(head, 0)
5944 end
5945
5946 function Babel.post_hyphenate_replace(head)
5947     Babel.hyphenate_replace(head, 1)
5948 end
5949
5950 function Babel.debug_hyph(w, wn, sc, first, last, last_match)
5951     local ss = ''
5952     for pp = 1, 40 do
5953         if wn[pp] then
5954             if wn[pp].id == 29 then
5955                 ss = ss .. unicode.utf8.char(wn[pp].char)
5956             else
5957                 ss = ss .. '{' .. wn[pp].id .. '}'
5958             end
5959         end
5960     end
5961     print('nod', ss)
5962     print('lst_m',
5963           string.rep(' ', unicode.utf8.len(
5964             string.sub(w, 1, last_match))-1) .. '>')
5965     print('str', w)
5966     print('sc', string.rep(' ', sc-1) .. '^')
5967     if first == last then

```

```

5968     print('f=l', string.rep(' ', first-1) .. '!')
5969 else
5970     print('f/l', string.rep(' ', first-1) .. '[' ..
5971         string.rep(' ', last-first-1) .. ']')
5972 end
5973 end
5974
5975 Babel.us_char = string.char(31)
5976
5977 function Babel.hyphenate_replace(head, mode)
5978     local u = unicode.utf8
5979     local lbkr = Babel.linebreaking.replacements[mode]
5980
5981     local word_head = head
5982
5983     while true do  %% for each subtext block
5984
5985         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
5986
5987         if Babel.debug then
5988             print()
5989             print((mode == 0) and '@@@<' or '@@@>', w)
5990         end
5991
5992         if nw == nil and w == '' then break end
5993
5994         if not lang then goto next end
5995         if not lbkr[lang] then goto next end
5996
5997         %% For each saved (pre|post)hyphenation. TODO. Reconsider how
5998         %% loops are nested.
5999         for k=1, #lbkr[lang] do
6000             local p = lbkr[lang][k].pattern
6001             local r = lbkr[lang][k].replace
6002
6003             if Babel.debug then
6004                 print('*****', p, mode)
6005             end
6006
6007             %% This variable is set in some cases below to the first *byte*
6008             %% after the match, either as found by u.match (faster) or the
6009             %% computed position based on sc if w has changed.
6010             local last_match = 0
6011             local step = 0
6012
6013             %% For every match.
6014             while true do
6015                 if Babel.debug then
6016                     print('====')
6017                 end
6018                 local new  %% used when inserting and removing nodes
6019
6020                 local matches = { u.match(w, p, last_match) }
6021
6022                 if #matches < 2 then break end
6023
6024                 %% Get and remove empty captures (with ())'s, which return a
6025                 %% number with the position), and keep actual captures
6026                 %% (from (...)), if any, in matches.

```



```

6027     local first = table.remove(matches, 1)
6028     local last  = table.remove(matches, #matches)
6029     %% Non re-fetched substrings may contain \31, which separates
6030     %% subsubstrings.
6031     if string.find(w:sub(first, last-1), Babel.us_char) then break end
6032
6033     local save_last = last %% with A()BC()D, points to D
6034
6035     %% Fix offsets, from bytes to unicode. Explained above.
6036     first = u.len(w:sub(1, first-1)) + 1
6037     last  = u.len(w:sub(1, last-1)) %% now last points to C
6038
6039     %% This loop stores in n small table the nodes
6040     %% corresponding to the pattern. Used by 'data' to provide a
6041     %% predictable behavior with 'insert' (now w_nodes is modified on
6042     %% the fly), and also access to 'remove'd nodes.
6043     local sc = first-1          %% Used below, too
6044     local data_nodes = {}
6045
6046     for q = 1, last-first+1 do
6047         data_nodes[q] = w_nodes[sc+q]
6048     end
6049
6050     %% This loop traverses the matched substring and takes the
6051     %% corresponding action stored in the replacement list.
6052     %% sc = the position in substr nodes / string
6053     %% rc = the replacement table index
6054     local rc = 0
6055
6056     while rc < last-first+1 do %% for each replacement
6057         if Babel.debug then
6058             print('.....', rc + 1)
6059         end
6060         sc = sc + 1
6061         rc = rc + 1
6062
6063         if Babel.debug then
6064             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6065             local ss = ''
6066             for itt in node.traverse(head) do
6067                 if itt.id == 29 then
6068                     ss = ss .. unicode.utf8.char(itt.char)
6069                 else
6070                     ss = ss .. '{' .. itt.id .. '}'
6071                 end
6072             end
6073             print('*****', ss)
6074         end
6075
6076         local crep = r[rc]
6077         local item = w_nodes[sc]
6078         local item_base = item
6079         local placeholder = Babel.us_char
6080         local d
6081
6082         if crep and crep.data then
6083             item_base = data_nodes[crep.data]
6084         end
6085

```

```

6086
6087     if crep then
6088         step = crep.step or 0
6089     end
6090
6091     if crep and next(crep) == nil then && = {}
6092         last_match = save_last    && Optimization
6093         goto next
6094
6095     elseif crep == nil or crep.remove then
6096         node.remove(head, item)
6097         table.remove(w_nodes, sc)
6098         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6099         sc = sc - 1    && Nothing has been inserted.
6100         last_match = utf8.offset(w, sc+1+step)
6101         goto next
6102
6103     elseif crep and crep.kashida then && Experimental
6104         node.set_attribute(item,
6105             luatexbase.registernumber'bblar@kashida',
6106             crep.kashida)
6107         last_match = utf8.offset(w, sc+1+step)
6108         goto next
6109
6110     elseif crep and crep.string then
6111         local str = crep.string(matches)
6112         if str == '' then && Gather with nil
6113             node.remove(head, item)
6114             table.remove(w_nodes, sc)
6115             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6116             sc = sc - 1    && Nothing has been inserted.
6117         else
6118             local loop_first = true
6119             for s in string.utfvalues(str) do
6120                 d = node.copy(item_base)
6121                 d.char = s
6122                 if loop_first then
6123                     loop_first = false
6124                     head, new = node.insert_before(head, item, d)
6125                     if sc == 1 then
6126                         word_head = head
6127                     end
6128                     w_nodes[sc] = d
6129                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6130                 else
6131                     sc = sc + 1
6132                     head, new = node.insert_before(head, item, d)
6133                     table.insert(w_nodes, sc, new)
6134                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6135                 end
6136                 if Babel.debug then
6137                     print('.....', 'str')
6138                     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6139                 end
6140             end && for
6141             node.remove(head, item)
6142         end && if ''
6143         last_match = utf8.offset(w, sc+1+step)
6144         goto next

```

```

6145
6146 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6147     d = node.new(7, 0)    %% (disc, discretionary)
6148     d.pre    = Babel.str_to_nodes(crep.pre, matches, item_base)
6149     d.post   = Babel.str_to_nodes(crep.post, matches, item_base)
6150     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6151     d.attr = item_base.attr
6152     if crep.pre == nil then    %% TeXbook p96
6153         d.penalty = crep.penalty or tex.hyphenpenalty
6154     else
6155         d.penalty = crep.penalty or tex.exhyphenpenalty
6156     end
6157     placeholder = '|'
6158     head, new = node.insert_before(head, item, d)
6159
6160 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6161     %% ERROR
6162
6163 elseif crep and crep.penalty then
6164     d = node.new(14, 0)    %% (penalty, userpenalty)
6165     d.attr = item_base.attr
6166     d.penalty = crep.penalty
6167     head, new = node.insert_before(head, item, d)
6168
6169 elseif crep and crep.space then
6170     %% 655360 = 10 pt = 10 * 65536 sp
6171     d = node.new(12, 13)    %% (glue, spaceskip)
6172     local quad = font.getfont(item_base.font).size or 655360
6173     node.setglue(d, crep.space[1] * quad,
6174                  crep.space[2] * quad,
6175                  crep.space[3] * quad)
6176     if mode == 0 then
6177         placeholder = ' '
6178     end
6179     head, new = node.insert_before(head, item, d)
6180
6181 elseif crep and crep.spacefactor then
6182     d = node.new(12, 13)    %% (glue, spaceskip)
6183     local base_font = font.getfont(item_base.font)
6184     node.setglue(d,
6185                  crep.spacefactor[1] * base_font.parameters['space'],
6186                  crep.spacefactor[2] * base_font.parameters['space_stretch'],
6187                  crep.spacefactor[3] * base_font.parameters['space_shrink'])
6188     if mode == 0 then
6189         placeholder = ' '
6190     end
6191     head, new = node.insert_before(head, item, d)
6192
6193 elseif mode == 0 and crep and crep.space then
6194     %% ERROR
6195
6196 end    %% ie replacement cases
6197
6198 %% Shared by disc, space and penalty.
6199 if sc == 1 then
6200     word_head = head
6201 end
6202 if crep.insert then
6203     w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)

```

```

6204         table.insert(w_nodes, sc, new)
6205         last = last + 1
6206     else
6207         w_nodes[sc] = d
6208         node.remove(head, item)
6209         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6210     end
6211
6212     last_match = utf8.offset(w, sc+1+step)
6213
6214     ::next::
6215
6216     end %% for each replacement
6217
6218     if Babel.debug then
6219         print('.....', '/')
6220         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6221     end
6222
6223     end %% for match
6224
6225     end %% for patterns
6226
6227     ::next::
6228     word_head = nw
6229 end %% for substring
6230 return head
6231 end
6232
6233 %% This table stores capture maps, numbered consecutively
6234 Babel.capture_maps = {}
6235
6236 %% The following functions belong to the next macro
6237 function Babel.capture_func(key, cap)
6238     local ret = "[" .. cap:gsub('{([0-9])}', "].m[%1]..[" .. "]"
6239     local cnt
6240     local u = unicode.utf8
6241     ret, cnt = ret:gsub('{([0-9])|([^|]+)|(.-)}', Babel.capture_func_map)
6242     if cnt == 0 then
6243         ret = u.gsub(ret, '{(%x%x%x%x+)}',
6244             function (n)
6245                 return u.char(tonumber(n, 16))
6246             end)
6247     end
6248     ret = ret:gsub("%[%[%]]%.", '')
6249     ret = ret:gsub("%.%[%[%]]%", '')
6250     return key .. "[=function(m) return ]] .. ret .. [[ end]]
6251 end
6252
6253 function Babel.capt_map(from, mapno)
6254     return Babel.capture_maps[mapno][from] or from
6255 end
6256
6257 %% Handle the {n|abc|ABC} syntax in captures
6258 function Babel.capture_func_map(capno, from, to)
6259     local u = unicode.utf8
6260     from = u.gsub(from, '{(%x%x%x%x+)}',
6261         function (n)
6262             return u.char(tonumber(n, 16))

```

```

6263     end)
6264     to = u.gsub(to, '{(%x%x%x%x+)}',
6265     function (n)
6266         return u.char(tonumber(n, 16))
6267     end)
6268     local froms = {}
6269     for s in string.utfcharacters(from) do
6270         table.insert(froms, s)
6271     end
6272     local cnt = 1
6273     table.insert(Babel.capture_maps, {})
6274     local mlen = table.getn(Babel.capture_maps)
6275     for s in string.utfcharacters(to) do
6276         Babel.capture_maps[mlen][froms[cnt]] = s
6277         cnt = cnt + 1
6278     end
6279     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
6280         (mlen) .. ").." .. "[["
6281 end
6282
6283 &% Create/Extend reversed sorted list of kashida weights:
6284 function Babel.capture_kashida(key, wt)
6285     wt = tonumber(wt)
6286     if Babel.kashida_wts then
6287         for p, q in ipairs(Babel.kashida_wts) do
6288             if wt == q then
6289                 break
6290             elseif wt > q then
6291                 table.insert(Babel.kashida_wts, p, wt)
6292                 break
6293             elseif table.getn(Babel.kashida_wts) == p then
6294                 table.insert(Babel.kashida_wts, wt)
6295             end
6296         end
6297     else
6298         Babel.kashida_wts = { wt }
6299     end
6300     return 'kashida = ' .. wt
6301 end
6302 }

```

Now the  $\TeX$  high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the  $\{n\}$  syntax. For example,  $\text{pre}=\{1\}\{1\}$ - 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).

```

6303 \catcode`\#=6
6304 \gdef\babelposthyphenation#1#2#3{&%
6305     \bbl@activateposthyphen
6306     \begingroup
6307         \def\babeltempa{\bbl@add@list\babeltempb}&%
6308         \let\babeltempb\@empty
6309         \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
6310         \bbl@replace\bbl@tempa{,}{ ,}&%
6311         \expandafter\bbl@foreach\expandafter{\bbl@tempa}&%

```

```

6312 \bbl@ifsamestring{##1}{remove}&%
6313 {\bbl@add@list\babeltempb{nil}}&%
6314 {\directlua{
6315     local rep = [=[#1]=]
6316     rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6317     rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
6318     rep = rep:gsub(' (no)%s*%s*([^\s,]*)', Babel.capture_func)
6319     rep = rep:gsub(' (pre)%s*%s*([^\s,]*)', Babel.capture_func)
6320     rep = rep:gsub(' (post)%s*%s*([^\s,]*)', Babel.capture_func)
6321     rep = rep:gsub('(string)%s*%s*([^\s,]*)', Babel.capture_func)
6322     tex.print([[\\string\babeltempa{}}] .. rep .. [[]]])
6323 }}&%
6324 \directlua{
6325     local lbkr = Babel.linebreaking.replacements[1]
6326     local u = unicode.utf8
6327     local id = \the\csname l@#1\endcsname
6328     &% Convert pattern:
6329     local patt = string.gsub(=[#2]=], '%s', '')
6330     if not u.find(patt, '()', nil, true) then
6331         patt = '()' .. patt .. '()'
6332     end
6333     patt = string.gsub(patt, '%(%)%^\', '^()')
6334     patt = string.gsub(patt, '%$$(%)', '()$')
6335     patt = u.gsub(patt, '{(.)}',
6336         function (n)
6337             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6338         end)
6339     patt = u.gsub(patt, '{(%x%x%x%x%x+)}',
6340         function (n)
6341             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6342         end)
6343     lbkr[id] = lbkr[id] or {}
6344     table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })
6345 }&%
6346 \endgroup}
6347 % TODO. Copypaste pattern.
6348 \gdef\babelprehyphenation#1#2#3{&%
6349 \bbl@activateprehyphen
6350 \begingroup
6351 \def\babeltempa{\bbl@add@list\babeltempb}&%
6352 \let\babeltempb\@empty
6353 \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
6354 \bbl@replace\bbl@tempa{,}{ ,}&%
6355 \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
6356 \bbl@ifsamestring{##1}{remove}&%
6357 {\bbl@add@list\babeltempb{nil}}&%
6358 {\directlua{
6359     local rep = [=[#1]=]
6360     rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6361     rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
6362     rep = rep:gsub('(string)%s*%s*([^\s,]*)', Babel.capture_func)
6363     rep = rep:gsub('(space)%s*%s*([^\s,]*)%s*([^\s,]*)%s*([^\s,]*)',
6364         'space = {' .. '%2, %3, %4' .. '}')
6365     rep = rep:gsub('(spacefactor)%s*%s*([^\s,]*)%s*([^\s,]*)%s*([^\s,]*)',
6366         'spacefactor = {' .. '%2, %3, %4' .. '}')
6367     rep = rep:gsub('(kashida)%s*%s*([^\s,]*)', Babel.capture_kashida)
6368     tex.print([[\\string\babeltempa{}}] .. rep .. [[]]])
6369 }}&%
6370 \directlua{

```

```

6371     local lbkr = Babel.linebreaking.replacements[0]
6372     local u = unicode.utf8
6373     local id = \the\csname bbl@id@@#1\endcsname
6374     &% Convert pattern:
6375     local patt = string.gsub(#[#2]=], '%s', '')
6376     local patt = string.gsub(patt, '|', ' ')
6377     if not u.find(patt, '()', nil, true) then
6378         patt = '()' .. patt .. '()'
6379     end
6380     &% patt = string.gsub(patt, '%(%)^', '^()')
6381     &% patt = string.gsub(patt, '([%^])%$%(%)', '%1()$')
6382     patt = u.gsub(patt, '{(.)}',
6383         function (n)
6384             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6385         end)
6386     patt = u.gsub(patt, '{(%x%x%x%x+)}',
6387         function (n)
6388             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6389         end)
6390     lbkr[id] = lbkr[id] or {}
6391     table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })
6392 }&%
6393 \endgroup}
6394 \endgroup
6395 \def\bbl@activateposthyphen{%
6396 \let\bbl@activateposthyphen\relax
6397 \directlua{
6398     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6399 }}
6400 \def\bbl@activateprehyphen{%
6401 \let\bbl@activateprehyphen\relax
6402 \directlua{
6403     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6404 }}

```

## 13.9 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`, `hline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```

6405 \bbl@trace{Redefinitions for bidi layout}
6406 \ifx\@eqnnum\@undefined\else
6407 \ifx\bbl@attr@dir\@undefined\else
6408 \edef\@eqnnum{%
6409     \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
6410     \unexpanded\expandafter{\@eqnnum}}
6411 \fi
6412 \fi
6413 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout

```

```

6414 \ifnum\bb1@bidimode>\z@
6415 \def\bb1@nextfake#1{% non-local changes, use always inside a group!
6416 \bb1@exp{%
6417 \mathdir\the\bodydir
6418 #1% Once entered in math, set boxes to restore values
6419 \<ifmode>%
6420 \everyvbox{%
6421 \the\everyvbox
6422 \bodydir\the\bodydir
6423 \mathdir\the\mathdir
6424 \everyhbox{\the\everyhbox}%
6425 \everyvbox{\the\everyvbox}}%
6426 \everyhbox{%
6427 \the\everyhbox
6428 \bodydir\the\bodydir
6429 \mathdir\the\mathdir
6430 \everyhbox{\the\everyhbox}%
6431 \everyvbox{\the\everyvbox}}%
6432 \<fi>}}%
6433 \def\@hangfrom#1{%
6434 \setbox\@tempboxa\hbox{{#1}}%
6435 \hangindent\wd\@tempboxa
6436 \ifnum\bb1@getluadir{page}=\bb1@getluadir{par}\else
6437 \shapemode\@ne
6438 \fi
6439 \noindent\box\@tempboxa}
6440 \fi
6441 \IfBabelLayout{tabular}
6442 {\let\bb1@OL@tabular\@tabular
6443 \bb1@replace\@tabular{$}\{\bb1@nextfake$}%
6444 \let\bb1@NL@tabular\@tabular
6445 \AtBeginDocument{%
6446 \ifx\bb1@NL@tabular\@tabular\else
6447 \bb1@replace\@tabular{$}\{\bb1@nextfake$}%
6448 \let\bb1@NL@tabular\@tabular
6449 \fi}}
6450 {}
6451 \IfBabelLayout{lists}
6452 {\let\bb1@OL@list\list
6453 \bb1@sreplace\list{\parshape}\{\bb1@listparshape}%
6454 \let\bb1@NL@list\list
6455 \def\bb1@listparshape#1#2#3{%
6456 \parshape #1 #2 #3 %
6457 \ifnum\bb1@getluadir{page}=\bb1@getluadir{par}\else
6458 \shapemode\tw@
6459 \fi}}
6460 {}
6461 \IfBabelLayout{graphics}
6462 {\let\bb1@pictresetdir\relax
6463 \def\bb1@pictsetdir#1{%
6464 \ifcase\bb1@thetextdir
6465 \let\bb1@pictresetdir\relax
6466 \else
6467 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6468 \or\textdir TLT
6469 \else\bodydir TLT \textdir TLT
6470 \fi
6471 % \text\par\dir required in pgf:
6472 \def\bb1@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%

```



```

6473 \fi}%
6474 \ifx\AddToHook\@undefined\else
6475 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6476 \directlua{
6477   Babel.get_picture_dir = true
6478   Babel.picture_has_bidi = 0
6479   function Babel.picture_dir (head)
6480     if not Babel.get_picture_dir then return head end
6481     for item in node.traverse(head) do
6482       if item.id == node.id'glyph' then
6483         local itemchar = item.char
6484         % TODO. Copypaste pattern from Babel.bidi (-r)
6485         local chardata = Babel.characters[itemchar]
6486         local dir = chardata and chardata.d or nil
6487         if not dir then
6488           for nn, et in ipairs(Babel.ranges) do
6489             if itemchar < et[1] then
6490               break
6491             elseif itemchar <= et[2] then
6492               dir = et[3]
6493               break
6494             end
6495           end
6496         end
6497         if dir and (dir == 'al' or dir == 'r') then
6498           Babel.picture_has_bidi = 1
6499         end
6500       end
6501     end
6502     return head
6503   end
6504   luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6505     "Babel.picture_dir")
6506 }%
6507 \AtBeginDocument{%
6508 \long\def\put(#1,#2)#3{%
6509 \killglue
6510 % Try:
6511 \ifx\bbl@pictresetdir\relax
6512 \def\bbl@tempc{0}%
6513 \else
6514 \directlua{
6515   Babel.get_picture_dir = true
6516   Babel.picture_has_bidi = 0
6517 }%
6518 \setbox\z@\hb@xt@\z@{%
6519 \@defaultunitsset\@tempdimc{#1}\unitlength
6520 \kern\@tempdimc
6521 #3\hss}%
6522 \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6523 \fi
6524 % Do:
6525 \@defaultunitsset\@tempdimc{#2}\unitlength
6526 \raise\@tempdimc\hb@xt@\z@{%
6527 \@defaultunitsset\@tempdimc{#1}\unitlength
6528 \kern\@tempdimc
6529 {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
6530 \ignorespaces}%
6531 \MakeRobust\put}%

```

```

6532 \fi
6533 \AtBeginDocument
6534 {\ifx\tikz@atbegin@node\undefined\else
6535   \ifx\AddToHook\undefined\else % TODO. Still tentative.
6536     \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6537     \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6538   \fi
6539   \let\bbl@OL@pgfpicture\pgfpicture
6540   \bbl@sreplace\pgfpicture{\pgfpicturetrue}%
6541   {\bbl@pictsetdir\z@\pgfpicturetrue}%
6542   \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6543   \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6544   \bbl@sreplace\tikz{\beginpgfgroup}%
6545   {\beginpgfgroup\bbl@pictsetdir\tw@}%
6546 \fi
6547 \ifx\AddToHook\undefined\else
6548   \AddToHook{env/tcolorbox/begin}{\bbl@pictsetdir\@ne}%
6549 \fi
6550 }}
6551 {}

```

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.

```

6552 \IfBabelLayout{counters}%
6553 {\let\bbl@OL@textsuperscript\textsuperscript
6554   \bbl@sreplace\textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6555   \let\bbl@latinarabic\@arabic
6556   \let\bbl@OL@arabic\@arabic
6557   \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
6558   \ifpackagewith{babel}{bidi=default}%
6559     {\let\bbl@asciroman=\@roman
6560      \let\bbl@OL@roman\@roman
6561      \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
6562      \let\bbl@asciiRoman=\@Roman
6563      \let\bbl@OL@roman\@Roman
6564      \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6565      \let\bbl@OL@labelenumii\labelenumii
6566      \def\labelenumii{}\theenumii}%
6567      \let\bbl@OL@p@enumiii\p@enumiii
6568      \def\p@enumiii{\p@enumii}\theenumii{}\}}{}
6569 \langle Footnote changes \rangle
6570 \IfBabelLayout{footnotes}%
6571 {\let\bbl@OL@footnote\footnote
6572   \BabelFootnote\footnote\language\@language}%
6573   \BabelFootnote\localfootnote\language\@language}%
6574   \BabelFootnote\mainfootnote{}\}}{}
6575 {}

```

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.

```

6576 \IfBabelLayout{extras}%
6577 {\let\bbl@OL@underline\underline
6578   \bbl@sreplace\underline{\$@\underline}{\bbl@nextfake\$@\underline}%
6579   \let\bbl@OL@LaTeX2e\LaTeX2e
6580   \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6581     \if b\expandafter\@car\@series\@nil\boldmath\fi
6582     \babelsublr}%
6583     \LaTeX\kern.15em\bbl@nextfake$_{\textstyle\varepsilon}$}}

```

```
6584 {}
6585 </luatex>
```

### 13.10 Auto bidi with basic and basic-r

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

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

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

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

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

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, *what* they do and *why*, and not only *how*), but I think (or I hope) I've managed to understand them. In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually *two* R modes (set explicitly in Unicode with RLM and ALM). In babel the dir is set by a higher protocol based on the language/script, which in turn sets the correct dir (<l>, <r> or <al>).

From UAX#9: "Where available, markup should be used instead of the explicit formatting characters". So, this simple version just ignores formatting characters. Actually, most of that annex is devoted to how to handle them.

BD14-BD16 are not implemented. Unicode (and the W3C) are making a great effort to deal with some special problematic cases in "streamed" plain text. I don't think this is the way to go – particular issues should be fixed by a high level interface taking into account the needs of the document. And here is where `luatex` excels, because everything related to bidi writing is under our control.

```
6586 (*basic-r)
6587 Babel = Babel or {}
6588
6589 Babel.bidi_enabled = true
6590
6591 require('babel-data-bidi.lua')
6592
6593 local characters = Babel.characters
6594 local ranges = Babel.ranges
6595
6596 local DIR = node.id("dir")
6597
6598 local function dir_mark(head, from, to, outer)
6599   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6600   local d = node.new(DIR)
6601   d.dir = '+' .. dir
6602   node.insert_before(head, from, d)
```

```

6603 d = node.new(DIR)
6604 d.dir = '-' .. dir
6605 node.insert_after(head, to, d)
6606 end
6607
6608 function Babel.bidi(head, ispar)
6609   local first_n, last_n      -- first and last char with nums
6610   local last_es              -- an auxiliary 'last' used with nums
6611   local first_d, last_d      -- first and last char in L/R block
6612   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):

```

6613   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6614   local strong_lr = (strong == 'l') and 'l' or 'r'
6615   local outer = strong
6616
6617   local new_dir = false
6618   local first_dir = false
6619   local inmath = false
6620
6621   local last_lr
6622
6623   local type_n = ''
6624
6625   for item in node.traverse(head) do
6626
6627     -- three cases: glyph, dir, otherwise
6628     if item.id == node.id'glyph'
6629       or (item.id == 7 and item.subtype == 2) then
6630
6631       local itemchar
6632       if item.id == 7 and item.subtype == 2 then
6633         itemchar = item.replace.char
6634       else
6635         itemchar = item.char
6636       end
6637       local chardata = characters[itemchar]
6638       dir = chardata and chardata.d or nil
6639       if not dir then
6640         for nn, et in ipairs(ranges) do
6641           if itemchar < et[1] then
6642             break
6643           elseif itemchar <= et[2] then
6644             dir = et[3]
6645             break
6646           end
6647         end
6648       end
6649       dir = dir or 'l'
6650       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).

```

6651   if new_dir then

```

```

6652     attr_dir = 0
6653     for at in node.traverse(item.attr) do
6654         if at.number == luatexbase.registernumber'bbl@attr@dir' then
6655             attr_dir = at.value % 3
6656         end
6657     end
6658     if attr_dir == 1 then
6659         strong = 'r'
6660     elseif attr_dir == 2 then
6661         strong = 'al'
6662     else
6663         strong = 'l'
6664     end
6665     strong_lr = (strong == 'l') and 'l' or 'r'
6666     outer = strong_lr
6667     new_dir = false
6668 end
6669
6670 if dir == 'nsm' then dir = strong end -- W1

```

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

```

6671     dir_real = dir -- We need dir_real to set strong below
6672     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:

```

6673     if strong == 'al' then
6674         if dir == 'en' then dir = 'an' end -- W2
6675         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6676         strong_lr = 'r' -- W3
6677     end

```

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

```

6678     elseif item.id == node.id'dir' and not inmath then
6679         new_dir = true
6680         dir = nil
6681     elseif item.id == node.id'math' then
6682         inmath = (item.subtype == 0)
6683     else
6684         dir = nil -- Not a char
6685     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>.

```

6686     if dir == 'en' or dir == 'an' or dir == 'et' then
6687         if dir ~= 'et' then
6688             type_n = dir
6689         end
6690         first_n = first_n or item
6691         last_n = last_es or item
6692         last_es = nil
6693     elseif dir == 'es' and last_n then -- W3+W6
6694         last_es = item
6695     elseif dir == 'cs' then -- it's right - do nothing
6696     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6697         if strong_lr == 'r' and type_n ~= '' then

```

```

6698     dir_mark(head, first_n, last_n, 'r')
6699     elseif strong_lr == 'l' and first_d and type_n == 'an' then
6700         dir_mark(head, first_n, last_n, 'r')
6701         dir_mark(head, first_d, last_d, outer)
6702         first_d, last_d = nil, nil
6703     elseif strong_lr == 'l' and type_n ~= '' then
6704         last_d = last_n
6705     end
6706     type_n = ''
6707     first_n, last_n = nil, nil
6708 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:

```

6709     if dir == 'l' or dir == 'r' then
6710         if dir ~= outer then
6711             first_d = first_d or item
6712             last_d = item
6713         elseif first_d and dir ~= strong_lr then
6714             dir_mark(head, first_d, last_d, outer)
6715             first_d, last_d = nil, nil
6716         end
6717     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.

```

6718     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6719         item.char = characters[item.char] and
6720             characters[item.char].m or item.char
6721     elseif (dir or new_dir) and last_lr ~= item then
6722         local mir = outer .. strong_lr .. (dir or outer)
6723         if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6724             for ch in node.traverse(node.next(last_lr)) do
6725                 if ch == item then break end
6726                 if ch.id == node.id'glyph' and characters[ch.char] then
6727                     ch.char = characters[ch.char].m or ch.char
6728                 end
6729             end
6730         end
6731     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).

```

6732     if dir == 'l' or dir == 'r' then
6733         last_lr = item
6734         strong = dir_real          -- Don't search back - best save now
6735         strong_lr = (strong == 'l') and 'l' or 'r'
6736     elseif new_dir then
6737         last_lr = nil
6738     end
6739 end

```

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

```

6740     if last_lr and outer == 'r' then

```

```

6741     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6742         if characters[ch.char] then
6743             ch.char = characters[ch.char].m or ch.char
6744         end
6745     end
6746 end
6747 if first_n then
6748     dir_mark(head, first_n, last_n, outer)
6749 end
6750 if first_d then
6751     dir_mark(head, first_d, last_d, outer)
6752 end

```

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

```

6753 return node.prev(head) or head
6754 end
6755 </basic-r>

```

And here the Lua code for bidi=basic:

```

6756 <(*basic>
6757 Babel = Babel or {}
6758
6759 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6760
6761 Babel.fontmap = Babel.fontmap or {}
6762 Babel.fontmap[0] = {}      -- l
6763 Babel.fontmap[1] = {}      -- r
6764 Babel.fontmap[2] = {}      -- al/an
6765
6766 Babel.bidi_enabled = true
6767 Babel.mirroring_enabled = true
6768
6769 require('babel-data-bidi.lua')
6770
6771 local characters = Babel.characters
6772 local ranges = Babel.ranges
6773
6774 local DIR = node.id('dir')
6775 local GLYPH = node.id('glyph')
6776
6777 local function insert_implicit(head, state, outer)
6778     local new_state = state
6779     if state.sim and state.eim and state.sim ~= state.eim then
6780         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6781         local d = node.new(DIR)
6782         d.dir = '+' .. dir
6783         node.insert_before(head, state.sim, d)
6784         local d = node.new(DIR)
6785         d.dir = '-' .. dir
6786         node.insert_after(head, state.eim, d)
6787     end
6788     new_state.sim, new_state.eim = nil, nil
6789     return head, new_state
6790 end
6791
6792 local function insert_numeric(head, state)
6793     local new
6794     local new_state = state

```

```

6795 if state.san and state.ean and state.san ~= state.ean then
6796     local d = node.new(DIR)
6797     d.dir = '+TLT'
6798     _, new = node.insert_before(head, state.san, d)
6799     if state.san == state.sim then state.sim = new end
6800     local d = node.new(DIR)
6801     d.dir = '-TLT'
6802     _, new = node.insert_after(head, state.ean, d)
6803     if state.ean == state.eim then state.eim = new end
6804 end
6805 new_state.san, new_state.ean = nil, nil
6806 return head, new_state
6807 end
6808
6809 -- TODO - \hbox with an explicit dir can lead to wrong results
6810 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6811 -- was s made to improve the situation, but the problem is the 3-dir
6812 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6813 -- well.
6814
6815 function Babel.bidi(head, ispar, hdir)
6816     local d -- d is used mainly for computations in a loop
6817     local prev_d = ''
6818     local new_d = false
6819
6820     local nodes = {}
6821     local outer_first = nil
6822     local inmath = false
6823
6824     local glue_d = nil
6825     local glue_i = nil
6826
6827     local has_en = false
6828     local first_et = nil
6829
6830     local ATDIR = luatexbase.registernumber'bbl@attr@dir'
6831
6832     local save_outer
6833     local temp = node.get_attribute(head, ATDIR)
6834     if temp then
6835         temp = temp % 3
6836         save_outer = (temp == 0 and 'l') or
6837             (temp == 1 and 'r') or
6838             (temp == 2 and 'al')
6839     elseif ispar then -- Or error? Shouldn't happen
6840         save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6841     else -- Or error? Shouldn't happen
6842         save_outer = ('TRT' == hdir) and 'r' or 'l'
6843     end
6844     -- when the callback is called, we are just _after_ the box,
6845     -- and the textdir is that of the surrounding text
6846     -- if not ispar and hdir ~= tex.textdir then
6847     --     save_outer = ('TRT' == hdir) and 'r' or 'l'
6848     -- end
6849     local outer = save_outer
6850     local last = outer
6851     -- 'al' is only taken into account in the first, current loop
6852     if save_outer == 'al' then save_outer = 'r' end
6853

```



```

6854 local fontmap = Babel.fontmap
6855
6856 for item in node.traverse(head) do
6857
6858     -- In what follows, #node is the last (previous) node, because the
6859     -- current one is not added until we start processing the neutrals.
6860
6861     -- three cases: glyph, dir, otherwise
6862     if item.id == GLYPH
6863         or (item.id == 7 and item.subtype == 2) then
6864
6865         local d_font = nil
6866         local item_r
6867         if item.id == 7 and item.subtype == 2 then
6868             item_r = item.replace -- automatic discs have just 1 glyph
6869         else
6870             item_r = item
6871         end
6872         local chardata = characters[item_r.char]
6873         d = chardata and chardata.d or nil
6874         if not d or d == 'nsm' then
6875             for nn, et in ipairs(ranges) do
6876                 if item_r.char < et[1] then
6877                     break
6878                 elseif item_r.char <= et[2] then
6879                     if not d then d = et[3]
6880                     elseif d == 'nsm' then d_font = et[3]
6881                     end
6882                     break
6883                 end
6884             end
6885         end
6886         d = d or 'l'
6887
6888         -- A short 'pause' in bidi for mapfont
6889         d_font = d_font or d
6890         d_font = (d_font == 'l' and 0) or
6891             (d_font == 'nsm' and 0) or
6892             (d_font == 'r' and 1) or
6893             (d_font == 'al' and 2) or
6894             (d_font == 'an' and 2) or nil
6895         if d_font and fontmap and fontmap[d_font][item_r.font] then
6896             item_r.font = fontmap[d_font][item_r.font]
6897         end
6898
6899         if new_d then
6900             table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6901             if inmath then
6902                 attr_d = 0
6903             else
6904                 attr_d = node.get_attribute(item, ATDIR)
6905                 attr_d = attr_d % 3
6906             end
6907             if attr_d == 1 then
6908                 outer_first = 'r'
6909                 last = 'r'
6910             elseif attr_d == 2 then
6911                 outer_first = 'r'
6912                 last = 'al'

```

```

6913         else
6914             outer_first = 'l'
6915             last = 'l'
6916         end
6917         outer = last
6918         has_en = false
6919         first_et = nil
6920         new_d = false
6921     end
6922
6923     if glue_d then
6924         if (d == 'l' and 'l' or 'r') ~= glue_d then
6925             table.insert(nodes, {glue_i, 'on', nil})
6926         end
6927         glue_d = nil
6928         glue_i = nil
6929     end
6930
6931     elseif item.id == DIR then
6932         d = nil
6933         new_d = true
6934
6935     elseif item.id == node.id'glue' and item.subtype == 13 then
6936         glue_d = d
6937         glue_i = item
6938         d = nil
6939
6940     elseif item.id == node.id'math' then
6941         inmath = (item.subtype == 0)
6942
6943     else
6944         d = nil
6945     end
6946
6947     -- AL <= EN/ET/ES      -- W2 + W3 + W6
6948     if last == 'al' and d == 'en' then
6949         d = 'an'          -- W3
6950     elseif last == 'al' and (d == 'et' or d == 'es') then
6951         d = 'on'          -- W6
6952     end
6953
6954     -- EN + CS/ES + EN      -- W4
6955     if d == 'en' and #nodes >= 2 then
6956         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6957             and nodes[#nodes-1][2] == 'en' then
6958             nodes[#nodes][2] = 'en'
6959         end
6960     end
6961
6962     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
6963     if d == 'an' and #nodes >= 2 then
6964         if (nodes[#nodes][2] == 'cs')
6965             and nodes[#nodes-1][2] == 'an' then
6966             nodes[#nodes][2] = 'an'
6967         end
6968     end
6969
6970     -- ET/EN                -- W5 + W7->1 / W6->on
6971     if d == 'et' then

```

```

6972     first_et = first_et or (#nodes + 1)
6973 elseif d == 'en' then
6974     has_en = true
6975     first_et = first_et or (#nodes + 1)
6976 elseif first_et then      -- d may be nil here !
6977     if has_en then
6978         if last == 'l' then
6979             temp = 'l'      -- W7
6980         else
6981             temp = 'en'     -- W5
6982         end
6983     else
6984         temp = 'on'        -- W6
6985     end
6986     for e = first_et, #nodes do
6987         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6988     end
6989     first_et = nil
6990     has_en = false
6991 end
6992
6993 -- Force mathdir in math if ON (currently works as expected only
6994 -- with 'l')
6995 if inmath and d == 'on' then
6996     d = ('TRT' == tex.mathdir) and 'r' or 'l'
6997 end
6998
6999 if d then
7000     if d == 'al' then
7001         d = 'r'
7002         last = 'al'
7003     elseif d == 'l' or d == 'r' then
7004         last = d
7005     end
7006     prev_d = d
7007     table.insert(nodes, {item, d, outer_first})
7008 end
7009
7010 outer_first = nil
7011
7012 end
7013
7014 -- TODO -- repeated here in case EN/ET is the last node. Find a
7015 -- better way of doing things:
7016 if first_et then      -- dir may be nil here !
7017     if has_en then
7018         if last == 'l' then
7019             temp = 'l'      -- W7
7020         else
7021             temp = 'en'     -- W5
7022         end
7023     else
7024         temp = 'on'        -- W6
7025     end
7026     for e = first_et, #nodes do
7027         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7028     end
7029 end
7030

```

```

7031 -- dummy node, to close things
7032 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7033
7034 ----- NEUTRAL -----
7035
7036 outer = save_outer
7037 last = outer
7038
7039 local first_on = nil
7040
7041 for q = 1, #nodes do
7042     local item
7043
7044     local outer_first = nodes[q][3]
7045     outer = outer_first or outer
7046     last = outer_first or last
7047
7048     local d = nodes[q][2]
7049     if d == 'an' or d == 'en' then d = 'r' end
7050     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7051
7052     if d == 'on' then
7053         first_on = first_on or q
7054     elseif first_on then
7055         if last == d then
7056             temp = d
7057         else
7058             temp = outer
7059         end
7060         for r = first_on, q - 1 do
7061             nodes[r][2] = temp
7062             item = nodes[r][1] -- MIRRORING
7063             if Babel.mirroring_enabled and item.id == GLYPH
7064                 and temp == 'r' and characters[item.char] then
7065                 local font_mode = font.fonts[item.font].properties.mode
7066                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
7067                     item.char = characters[item.char].m or item.char
7068                 end
7069             end
7070         end
7071         first_on = nil
7072     end
7073
7074     if d == 'r' or d == 'l' then last = d end
7075 end
7076
7077 ----- IMPLICIT, REORDER -----
7078
7079 outer = save_outer
7080 last = outer
7081
7082 local state = {}
7083 state.has_r = false
7084
7085 for q = 1, #nodes do
7086
7087     local item = nodes[q][1]
7088
7089     outer = nodes[q][3] or outer

```

```

7090
7091     local d = nodes[q][2]
7092
7093     if d == 'nsm' then d = last end           -- W1
7094     if d == 'en' then d = 'an' end
7095     local isdir = (d == 'r' or d == 'l')
7096
7097     if outer == 'l' and d == 'an' then
7098         state.san = state.san or item
7099         state.ean = item
7100     elseif state.san then
7101         head, state = insert_numeric(head, state)
7102     end
7103
7104     if outer == 'l' then
7105         if d == 'an' or d == 'r' then        -- im -> implicit
7106             if d == 'r' then state.has_r = true end
7107             state.sim = state.sim or item
7108             state.eim = item
7109         elseif d == 'l' and state.sim and state.has_r then
7110             head, state = insert_implicit(head, state, outer)
7111         elseif d == 'l' then
7112             state.sim, state.eim, state.has_r = nil, nil, false
7113         end
7114     else
7115         if d == 'an' or d == 'l' then
7116             if nodes[q][3] then -- nil except after an explicit dir
7117                 state.sim = item -- so we move sim 'inside' the group
7118             else
7119                 state.sim = state.sim or item
7120             end
7121             state.eim = item
7122         elseif d == 'r' and state.sim then
7123             head, state = insert_implicit(head, state, outer)
7124         elseif d == 'r' then
7125             state.sim, state.eim = nil, nil
7126         end
7127     end
7128
7129     if isdir then
7130         last = d           -- Don't search back - best save now
7131     elseif d == 'on' and state.san then
7132         state.san = state.san or item
7133         state.ean = item
7134     end
7135
7136 end
7137
7138 return node.prev(head) or head
7139 end
7140 </basic>

```

## 14 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.

## 15 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.

```
7141 ⟨*nil⟩
7142 \ProvidesLanguage{nil}[⟨⟨date⟩⟩]⟨⟨version⟩⟩ Nil language]
7143 \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.

```
7144 \ifx\l@nil\undefined
7145   \newlanguage\l@nil
7146   \@namedef{bbl@hyphendata@the\l@nil}{}{}{}% Remove warning
7147   \let\bbl@elt\relax
7148   \edef\bbl@languages{% Add it to the list of languages
7149     \bbl@languages\bbl@elt{nil}{the\l@nil}{}{}}
7150 \fi
```

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

```
7151 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
\datenil
7152 \let\captionnil\empty
7153 \let\datenil\empty
```

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

```
7154 \ldf@finish{nil}
7155 ⟨/nil⟩
```

## 16 Support for Plain T<sub>E</sub>X (plain.def)

### 16.1 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`.

```
7156 <(*bplain | blplain)
7157 \catcode`\{=1 % left brace is begin-group character
7158 \catcode`\}=2 % right brace is end-group character
7159 \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).

```
7160 \openin 0 hyphen.cfg
7161 \ifeof0
7162 \else
7163   \let\input
```

Then `\input` is defined to forget about its argument and load `hyphen.cfg` instead. Once that's done the original meaning of `\input` can be restored and the definition of `\a` can be forgotten.

```
7164   \def\input #1 {%
7165     \let\input\input
7166     \a hyphen.cfg
7167     \let\input\undefined
7168   }
7169 \fi
7170 </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`.

```
7171 <bplain>\a plain.tex
7172 <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.

```
7173 <bplain>\def\fmtname{babel-plain}
7174 <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.

## 16.2 Emulating some $\text{\LaTeX}$ features

The following code duplicates or emulates parts of  $\text{\LaTeX} 2_{\epsilon}$  that are needed for `babel`.

```
7175 <<(*Emulate LaTeX)>> ≡
7176 % == Code for plain ==
7177 \def\@empty{}
7178 \def\loadlocalcfg#1{%
7179   \openin0#1.cfg
7180   \ifeof0
7181     \closein0
7182   \else
7183     \closein0
7184     {\immediate\write16{*****}%
7185      \immediate\write16{* Local config file #1.cfg used}%
7186      \immediate\write16{*}%
7187     }
7188     \input #1.cfg\relax
7189   \fi
7190 \endofldef}
```

## 16.3 General tools

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

```

7191 \long\def\@firstofone#1{#1}
7192 \long\def\@firstoftwo#1#2{#1}
7193 \long\def\@secondoftwo#1#2{#2}
7194 \def\@nnil{\@nil}
7195 \def\@gobbletwo#1#2{}
7196 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7197 \def\@star@or@long#1{%
7198   \@ifstar
7199   {\let\l@ngrel@x\relax#1}%
7200   {\let\l@ngrel@x\long#1}}
7201 \let\l@ngrel@x\relax
7202 \def\@car#1#2\@nil{#1}
7203 \def\@cdr#1#2\@nil{#2}
7204 \let\@typeset@protect\relax
7205 \let\protected@edef\edef
7206 \long\def\@gobble#1{}
7207 \edef\@backslashchar{\expandafter\@gobble\string\}
7208 \def\strip@prefix#1>{}
7209 \def\g@addto@macro#1#2{%
7210   \toks@\expandafter{#1#2}%
7211   \xdef#1{\the\toks@}}
7212 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
7213 \def\@nameuse#1{\csname #1\endcsname}
7214 \def\@ifundefined#1{%
7215   \expandafter\ifx\csname#1\endcsname\relax
7216     \expandafter\@firstoftwo
7217   \else
7218     \expandafter\@secondoftwo
7219   \fi}
7220 \def\@expandtwoargs#1#2#3{%
7221   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7222 \def\zap@space#1 #2{%
7223   #1%
7224   \ifx#2\@empty\else\expandafter\zap@space\fi
7225   #2}
7226 \let\bbl@trace\@gobble

```

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

```

7227 \ifx\@preamblecmds\@undefined
7228   \def\@preamblecmds{}
7229 \fi
7230 \def\@onlypreamble#1{%
7231   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
7232     \@preamblecmds\do#1}}
7233 \@onlypreamble\@onlypreamble

```

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

```

7234 \def\begindocument{%
7235   \@begindocumenthook
7236   \global\let\@begindocumenthook\@undefined
7237   \def\do##1{\global\let##1\@undefined}%
7238   \@preamblecmds
7239   \global\let\do\noexpand}
7240 \ifx\@begindocumenthook\@undefined
7241   \def\@begindocumenthook{}

```



```

7242 \fi
7243 \@onlypreamble\@begindocumenthook
7244 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

We also have to mimick LATEX's \AtEndOfPackage. Our replacement macro is much simpler; it stores
its argument in \@endofldf.

7245 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
7246 \@onlypreamble\AtEndOfPackage
7247 \def\@endofldf{}
7248 \@onlypreamble\@endofldf
7249 \let\bbl@afterlang\@empty
7250 \chardef\bbl@opt@hyphenmap\z@

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

7251 \catcode`\&=\z@
7252 \ifx&\if@files\@undefined
7253   \expandafter\let\csname if@files\expandafter\endcsname
7254     \csname iffalse\endcsname
7255 \fi
7256 \catcode`\&=4

Mimick LATEX's commands to define control sequences.

7257 \def\newcommand{\@star@or@long\new@command}
7258 \def\new@command#1{%
7259   \@testopt{\@newcommand#1}0}
7260 \def\@newcommand#1[#2]{%
7261   \@ifnextchar [{\@xargdef#1[#2]}%
7262     {\@argdef#1[#2]}}
7263 \long\def\@argdef#1[#2]#3{%
7264   \@yargdef#1\@ne{#2}{#3}}
7265 \long\def\@xargdef#1[#2][#3]#4{%
7266   \expandafter\def\expandafter#1\expandafter{%
7267     \expandafter\@protected@testopt\expandafter #1%
7268     \csname\string#1\expandafter\endcsname{#3}}%
7269   \expandafter\@yargdef \csname\string#1\endcsname
7270   \tw@{#2}{#4}}
7271 \long\def\@yargdef#1#2#3{%
7272   \@tempcnta#3\relax
7273   \advance \@tempcnta \@ne
7274   \let\@hash@\relax
7275   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7276   \@tempcntb #2%
7277   \@whilenum\@tempcntb <\@tempcnta
7278   \do{%
7279     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
7280     \advance\@tempcntb \@ne}%
7281   \let\@hash@###
7282   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
7283 \def\providecommand{\@star@or@long\provide@command}
7284 \def\provide@command#1{%
7285   \begingroup
7286   \escapechar\m@ne\xdef\@tempa{\string#1}%
7287   \endgroup
7288   \expandafter\ifundefined\@tempa
7289     {\def\reserved@a{\new@command#1}}%
7290     {\let\reserved@a\relax
7291       \def\reserved@a{\new@command\reserved@a}}%
7292   \reserved@a}%

```

```

7293 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7294 \def\declare@robustcommand#1{%
7295   \edef\reserved@a{\string#1}%
7296   \def\reserved@b{#1}%
7297   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7298   \edef#1{%
7299     \ifx\reserved@a\reserved@b
7300       \noexpand\x@protect
7301       \noexpand#1%
7302     \fi
7303     \noexpand\protect
7304     \expandafter\noexpand\csname
7305       \expandafter\@gobble\string#1 \endcsname
7306   }%
7307   \expandafter\new@command\csname
7308     \expandafter\@gobble\string#1 \endcsname
7309 }
7310 \def\x@protect#1{%
7311   \ifx\protect\@typeset@protect\else
7312     \@x@protect#1%
7313   \fi
7314 }
7315 \catcode`\&=\z@ % Trick to hide conditionals
7316 \def\@x@protect#1&fi##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`.

```

7317 \def\bbl@tempa{\csname newif\endcsname&ifin@}
7318 \catcode`\&=4
7319 \ifx\in@\@undefined
7320   \def\in@#1#2{%
7321     \def\in@##1#1##2##3\in@{%
7322       \ifx\in@##2\in@false\else\in@true\fi}%
7323     \in@#2#1\in@\in@@}
7324 \else
7325   \let\bbl@tempa\@empty
7326 \fi
7327 \bbl@tempa

```

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

```

7328 \def\ifpackagewith#1#2#3#4{#3}

```

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

```

7329 \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  $\text{\LaTeX 2}_\epsilon$  versions; just enough to make things work in plain  $\text{\TeX}$  environments.

```

7330 \ifx\@tempcnta\@undefined
7331   \csname newcount\endcsname\@tempcnta\relax
7332 \fi
7333 \ifx\@tempcntb\@undefined
7334   \csname newcount\endcsname\@tempcntb\relax
7335 \fi

```

To prevent wasting two counters in L<sup>A</sup>T<sub>E</sub>X 2.09 (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```

7336 \ifx\bye\@undefined
7337   \advance\count10 by -2\relax
7338 \fi
7339 \ifx\@ifnextchar\@undefined
7340   \def\@ifnextchar#1#2#3{%
7341     \let\reserved@d=#1%
7342     \def\reserved@a{#2}\def\reserved@b{#3}%
7343     \futurelet\@let@token\@ifnch}
7344   \def\@ifnch{%
7345     \ifx\@let@token\@sptoken
7346       \let\reserved@c\@xifnch
7347     \else
7348       \ifx\@let@token\reserved@d
7349         \let\reserved@c\reserved@a
7350       \else
7351         \let\reserved@c\reserved@b
7352       \fi
7353     \fi
7354     \reserved@c}
7355   \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
7356   \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
7357 \fi
7358 \def\@testopt#1#2{%
7359   \@ifnextchar[#{1}{#1[#2]}}
7360 \def\@protected@testopt#1{%
7361   \ifx\protect\@typeset@protect
7362     \expandafter\@testopt
7363   \else
7364     \@x@protect#1%
7365   \fi}
7366 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
7367   #2\relax}\fi}
7368 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
7369   \else\expandafter\@gobble\fi{#1}}

```

## 16.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain T<sub>E</sub>X environment.

```

7370 \def\DeclareTextCommand{%
7371   \@dec@text@cmd\providecommand
7372 }
7373 \def\ProvideTextCommand{%
7374   \@dec@text@cmd\providecommand
7375 }
7376 \def\DeclareTextSymbol#1#2#3{%
7377   \@dec@text@cmd\chardef#1{#2}#3\relax
7378 }
7379 \def\@dec@text@cmd#1#2#3{%
7380   \expandafter\def\expandafter#2%
7381     \expandafter{%
7382       \csname#3-cmd\expandafter\endcsname
7383       \expandafter#2%
7384       \csname#3\string#2\endcsname
7385     }%
7386 %   \let\@ifdefinable\@rc@ifdefinable
7387   \expandafter#1\csname#3\string#2\endcsname

```

```

7388 }
7389 \def\@current@cmd#1{%
7390   \ifx\protect\@typeset@protect\else
7391     \noexpand#1\expandafter\@gobble
7392   \fi
7393 }
7394 \def\@changed@cmd#1#2{%
7395   \ifx\protect\@typeset@protect
7396     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
7397       \expandafter\ifx\csname ?\string#1\endcsname\relax
7398         \expandafter\def\csname ?\string#1\endcsname{%
7399           \@changed@x@err{#1}%
7400         }%
7401       \fi
7402     \global\expandafter\let
7403       \csname\cf@encoding\string#1\expandafter\endcsname
7404       \csname ?\string#1\endcsname
7405     \fi
7406     \csname\cf@encoding\string#1%
7407       \expandafter\endcsname
7408   \else
7409     \noexpand#1%
7410   \fi
7411 }
7412 \def\@changed@x@err#1{%
7413   \errhelp{Your command will be ignored, type <return> to proceed}%
7414   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}%
7415 \def\DeclareTextCommandDefault#1{%
7416   \DeclareTextCommand#1?%
7417 }
7418 \def\ProvideTextCommandDefault#1{%
7419   \ProvideTextCommand#1?%
7420 }
7421 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
7422 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
7423 \def\DeclareTextAccent#1#2#3{%
7424   \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
7425 }
7426 \def\DeclareTextCompositeCommand#1#2#3#4{%
7427   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
7428   \edef\reserved@b{\string##1}%
7429   \edef\reserved@c{%
7430     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
7431   \ifx\reserved@b\reserved@c
7432     \expandafter\expandafter\expandafter\ifx
7433       \expandafter\@car\reserved@a\relax\relax\@nil
7434     \@text@composite
7435   \else
7436     \edef\reserved@b##1{%
7437       \def\expandafter\noexpand
7438         \csname#2\string#1\endcsname###1{%
7439           \noexpand\@text@composite
7440             \expandafter\noexpand\csname#2\string#1\endcsname
7441             ###1\noexpand\empty\noexpand\@text@composite
7442             {##1}%
7443         }%
7444     }%
7445     \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
7446   \fi

```

```

7447 \expandafter\def\csname\expandafter\string\csname
7448 #2\endcsname\string#1-\string#3\endcsname{#4}
7449 \else
7450 \errhelp{Your command will be ignored, type <return> to proceed}%
7451 \errmessage{\string\DeclareTextCompositeCommand\space used on
7452 inappropriate command \protect#1}
7453 \fi
7454 }
7455 \def\@text@composite#1#2#3\@text@composite{%
7456 \expandafter\@text@composite@x
7457 \csname\string#1-\string#2\endcsname
7458 }
7459 \def\@text@composite@x#1#2{%
7460 \ifx#1\relax
7461 #2%
7462 \else
7463 #1%
7464 \fi
7465 }
7466 %
7467 \def\@strip@args#1:#2-#3\@strip@args{#2}
7468 \def\DeclareTextComposite#1#2#3#4{%
7469 \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
7470 \bgroup
7471 \lccode`\@=#4%
7472 \lowercase{%
7473 \egroup
7474 \reserved@a @%
7475 }%
7476 }
7477 %
7478 \def\UseTextSymbol#1#2{#2}
7479 \def\UseTextAccent#1#2#3{}
7480 \def\@use@text@encoding#1{}
7481 \def\DeclareTextSymbolDefault#1#2{%
7482 \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
7483 }
7484 \def\DeclareTextAccentDefault#1#2{%
7485 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
7486 }
7487 \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.

```

7488 \DeclareTextAccent{"}{OT1}{127}
7489 \DeclareTextAccent{'}{OT1}{19}
7490 \DeclareTextAccent{^}{OT1}{94}
7491 \DeclareTextAccent`}{OT1}{18}
7492 \DeclareTextAccent{~}{OT1}{126}

```

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

```

7493 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
7494 \DeclareTextSymbol{\textquotedblright}{OT1}{93}
7495 \DeclareTextSymbol{\textquoteleft}{OT1}{96}
7496 \DeclareTextSymbol{\textquoteright}{OT1}{97}
7497 \DeclareTextSymbol{\i}{OT1}{16}
7498 \DeclareTextSymbol{\ss}{OT1}{25}

```

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

```

7499 \ifx\scriptsize\@undefined
7500 \let\scriptsize\sevenrm
7501 \fi
7502 % End of code for plain
7503 <</Emulate LaTeX>>

A proxy file:
7504 <plain>
7505 \input babel.def
7506 </plain>

```

## 17 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] Hubert Partl, *German  $\text{\TeX}$* , *TUGboat* 9 (1988) #1, p. 70–72.
- [10] Joachim Schrod, *International  $\text{\LaTeX}$  is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [11] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using  $\text{\LaTeX}$* , Springer, 2002, p. 301–373.
- [12] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).