

# Babel

Version 3.16

2018/01/04

*Original author*

Johannes L. Braams

*Current maintainer*

Javier Bezos

The standard distribution of  $\LaTeX$  contains a number of document classes that are meant to be used, but also serve as examples for other users to create their own document classes. These document classes have become very popular among  $\LaTeX$  users. But it should be kept in mind that they were designed for American tastes and typography. At one time they even contained a number of hard-wired texts.

This manual describes babel, a package that makes use of the capabilities of  $\TeX$  version 3 and, to some extent, xetex and luatex, to provide an environment in which documents can be typeset in a language other than US English, or in more than one language or script.

Current development is focused on Unicode engines (Xe $\TeX$  and Lua $\TeX$ ) and the so-called *complex scripts*. New features related to font selection, bidi writing and the like will be added incrementally.

Babel provides support (total or partial) for about 200 languages, either as a “classical” package option or as an `ini` file. Furthermore, new languages can be created from scratch easily.

# 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 . . . . .	5
1.3	Modifiers . . . . .	6
1.4	xelatex and lualatex . . . . .	6
1.5	Troubleshooting . . . . .	7
1.6	Plain . . . . .	8
1.7	Basic language selectors . . . . .	8
1.8	Auxiliary language selectors . . . . .	9
1.9	More on selection . . . . .	9
1.10	Shorthands . . . . .	10
1.11	Package options . . . . .	14
1.12	The base option . . . . .	16
1.13	ini files . . . . .	16
1.14	Selecting fonts . . . . .	22
1.15	Modifying a language . . . . .	24
1.16	Creating a language . . . . .	24
1.17	Getting the current language name . . . . .	26
1.18	Hyphenation tools . . . . .	27
1.19	Selecting scripts . . . . .	28
1.20	Selecting directions . . . . .	29
1.21	Language attributes . . . . .	30
1.22	Hooks . . . . .	30
1.23	Languages supported by babel . . . . .	31
1.24	Tips, workarounds, know issues and notes . . . . .	33
1.25	Current and future work . . . . .	34
1.26	Tentative and experimental code . . . . .	35
<b>2</b>	<b>Loading languages with language.dat</b>	<b>35</b>
2.1	Format . . . . .	36
<b>3</b>	<b>The interface between the core of babel and the language definition files</b>	<b>37</b>
3.1	Guidelines for contributed languages . . . . .	38
3.2	Basic macros . . . . .	38
3.3	Skeleton . . . . .	39
3.4	Support for active characters . . . . .	40
3.5	Support for saving macro definitions . . . . .	41
3.6	Support for extending macros . . . . .	41
3.7	Macros common to a number of languages . . . . .	41
3.8	Encoding-dependent strings . . . . .	42
<b>4</b>	<b>Changes</b>	<b>45</b>
4.1	Changes in babel version 3.9 . . . . .	45
4.2	Changes in babel version 3.7 . . . . .	46
<b>II</b>	<b>The code</b>	<b>46</b>
<b>5</b>	<b>Identification and loading of required files</b>	<b>46</b>

<b>6</b>	<b>Tools</b>	<b>47</b>
6.1	Multiple languages . . . . .	50
<b>7</b>	<b>The Package File (<math>\LaTeX</math>, babel.sty)</b>	<b>51</b>
7.1	base . . . . .	52
7.2	key=value options and other general option . . . . .	53
7.3	Conditional loading of shorthands . . . . .	54
7.4	Language options . . . . .	56
<b>8</b>	<b>The kernel of Babel (babel.def, common)</b>	<b>58</b>
8.1	Tools . . . . .	58
8.2	Hooks . . . . .	60
8.3	Setting up language files . . . . .	62
8.4	Shorthands . . . . .	64
8.5	Language attributes . . . . .	73
8.6	Support for saving macro definitions . . . . .	75
8.7	Short tags . . . . .	76
8.8	Hyphens . . . . .	76
8.9	Multiencoding strings . . . . .	78
8.10	Macros common to a number of languages . . . . .	84
8.11	Making glyphs available . . . . .	84
8.11.1	Quotation marks . . . . .	84
8.11.2	Letters . . . . .	85
8.11.3	Shorthands for quotation marks . . . . .	86
8.11.4	Umlauts and tremas . . . . .	87
8.12	Layout . . . . .	89
<b>9</b>	<b>The kernel of Babel (babel.def, only <math>\LaTeX</math>)</b>	<b>89</b>
9.1	The redefinition of the style commands . . . . .	89
9.2	Creating languages . . . . .	90
9.3	Cross referencing macros . . . . .	95
9.4	Marks . . . . .	99
9.5	Preventing clashes with other packages . . . . .	100
9.5.1	ifthen . . . . .	100
9.5.2	varioref . . . . .	101
9.5.3	hhline . . . . .	101
9.5.4	hyperref . . . . .	101
9.5.5	fancyhdr . . . . .	102
9.6	Encoding and fonts . . . . .	102
9.7	Basic bidi support . . . . .	104
9.8	Local Language Configuration . . . . .	106
<b>10</b>	<b>Multiple languages (switch.def)</b>	<b>107</b>
10.1	Selecting the language . . . . .	108
10.2	Errors . . . . .	115
<b>11</b>	<b>Loading hyphenation patterns</b>	<b>117</b>
<b>12</b>	<b>Font handling with fontspec</b>	<b>122</b>
<b>13</b>	<b>Hooks for XeTeX and LuaTeX</b>	<b>125</b>
13.1	XeTeX . . . . .	125
13.2	Layout . . . . .	126
13.3	LuaTeX . . . . .	128
13.4	Layout . . . . .	134
13.5	Auto bidi with basic-r . . . . .	135

<b>14</b>	<b>The ‘nil’ language</b>	<b>139</b>
<b>15</b>	<b>Support for Plain T<sub>E</sub>X (plain.def)</b>	<b>139</b>
15.1	Not renaming hyphen.tex . . . . .	139
15.2	Emulating some L <sup>A</sup> T <sub>E</sub> X features . . . . .	140
15.3	General tools . . . . .	141
15.4	Encoding related macros . . . . .	144
15.5	Babel options . . . . .	147
<b>16</b>	<b>Acknowledgements</b>	<b>148</b>

# Part I

## User guide

- This user guide focuses on  $\LaTeX$ . There are also some notes on its use with Plain  $\TeX$ .
- Changes and new features with relation to version 3.8 are highlighted with **New X.XX**. The most recent features could be still unstable. Please, report any issues you find.
- If you are interested in the  $\TeX$  multilingual support, please join the kadingira list on <http://tug.org/mailman/listinfo/kadingira>. You can follow the development of babel on <https://github.com/latex3/latex2e/tree/master/required/babel> (which provides some sample files, too).
- See section 3.1 for contributing a language.

## 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.

**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:

```
\documentclass{article}

\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}

\usepackage[french]{babel}

\begin{document}

Plus ça change, plus c'est la même chose!

\end{document}
```

**WARNING** A common source of trouble is a wrong setting of the input encoding. Make sure you set the encoding actually used by your editor.

Another approach is making the language (french in the example) a global option in order to let other packages detect and use it:

```
\documentclass[french]{article}
\usepackage{babel}
\usepackage{varioref}
```

In this last example, the package `varioref` will also see the option and will be able to use it.

**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.

## 1.2 Multilingual documents

In multilingual documents, just use several options. The last one is considered the main language, activated by default. Sometimes, the main language changes the document layout (eg, spanish and french).

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

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

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

You can also set the main language explicitly:

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

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

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

**WARNING** In the preamble the main language has *not* been selected, except hyphenation patterns and the name assigned to `\languagename` (in particular, shorthands, captions and date are not activated). If you need to define boxes and the like in the preamble, you might want to use some of the language selectors described below.

To switch the language there are two basic macros, described below in detail:

`\selectlanguage` is used for blocks of text, while `\foreignlanguage` is for chunks of text inside paragraphs.

**EXAMPLE** A full bilingual document follows. The main language is french, which is activated when the document begins.

```
\documentclass{article}

\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}

\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}

```

### 1.3 Modifiers

**New 3.9c** The basic behaviour 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 accept 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 is a more general mechanism.

### 1.4 xelatex and luatex

Many languages are compatible with xetex and luatex. With them you can use babel to localize the documents.

The Latin script is covered by default in current L<sup>A</sup>T<sub>E</sub>X (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** The following bilingual, single script document in UTF-8 encoding just prints a couple of ‘captions’ and `\today` in Danish and Vietnamese. No additional packages are required.

```

\documentclass{article}

\usepackage[vietnamese,danish]{babel}

\begin{document}

\prefacename{} -- \alsoname{} -- \today

\selectlanguage{vietnamese}

\prefacename{} -- \alsoname{} -- \today

\end{document}

```

**EXAMPLE** Here is a simple monolingual document in Russian (text from the Wikipedia). Note neither `fontenc` nor `inputenc` are necessary, but the document should be encoded

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

in UTF-8 and a so-called Unicode font must be loaded (in this example `\babelfont` is used, described below).

```
\documentclass{article}

\usepackage[russian]{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}
```

## 1.5 Troubleshooting

- Loading directly sty files in  $\LaTeX$  (ie, `\usepackage{<language>}`) is deprecated and you will get the error:<sup>2</sup>

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

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

```
! Package babel Error: Unknown language `LANG'. Either you have misspelled
(babel)                its name, it has not been installed, or you requested
(babel)                it in a previous run. Fix its name, install it or just
(babel)                rerun the file, respectively
```

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.

- 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.

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

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

## 1.6 Plain

In 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 Plain.<sup>4</sup>

## 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.

**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`  $\langle language \rangle \langle text \rangle$

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).

---

<sup>4</sup>Even in the babel kernel there were some macros not compatible with plain. Hopefully these issues will be fixed soon.

## 1.8 Auxiliary language selectors

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

The environment `otherlanguage` does basically the same as `\selectlanguage`, except the 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*}`  $\langle\text{language}\rangle$  ... `\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 behaviour and it is just a version as environment of `\foreignlanguage`.

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

The environment `hyphenrules` can be used to select *only* the hyphenation rules to be used (it can be used as command, too). This can for instance be used to select ‘nohyphenation’, provided that in `language.dat` the ‘language’ nohyphenation is defined by loading `zerohyph.tex`. It deactivates language shorthands, too (but not user shorthands). Except for these simple uses, `hyphenrules` is discouraged and `otherlanguage*` (the starred version) is preferred, as the former does not take into account possible changes in encodings of characters like, say, ‘ done by some languages (eg, italian, french, ukraineb). To set hyphenation exceptions, use `\babelhyphenation` (see below).

## 1.9 More on selection

`\babeltags`  $\langle\text{tag1}\rangle = \langle\text{language1}\rangle, \langle\text{tag2}\rangle = \langle\text{language2}\rangle, \dots$

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

**EXAMPLE** With

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

you can write

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

and

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

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

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

`\babelensure` [`include=<commands>`], [`exclude=<commands>`], [`fontenc=<encoding>`] {<language>}

**New 3.9i** Except in a few languages, like 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 `fontenc`.<sup>5</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, as for example: (1) in some languages shorthands such as "a are defined to be able to hyphenate the word if the encoding is OT1;

<sup>5</sup>With it encoded string may not work as expected.

(2) in some languages shorthands such as ! are used to insert the right amount of white space; (3) several kinds of discretionary and breaks can be inserted easily with "-", "=", etc. The package inputenc as well as xetex and luatex have alleviated entering non-ASCII characters, but minority languages and some kinds of text can still require characters not directly available on the keyboards (and sometimes not even as separated or precomposed Unicode characters). As to the point 2, now pdfTeX provides \kernbcode, and luatex can manipulate the glyph list. Tools for point 3 can be still very useful in general. There are three levels of shorthands: *user*, *language*, and *system* (by order of precedence). Version 3.9 introduces the *language user* level on top of the user level, as described below. In most cases, you will use only shorthands provided by languages.

**NOTE** Note 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) 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 it is deactivated with, eg, string).

A typical error when using shorthands is the following:

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

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

```
\shorthandon  {\shorthands-list}
\shorthandoff *{\shorthands-list}
```

It is sometimes necessary to switch a shorthand character off temporarily, because it must be used in an entirely different way. For this purpose, the user commands \shorthandoff and \shorthandon are provided. They each take a list of characters as their arguments. The command \shorthandoff sets the \catcode for each of the characters in its argument to other (12); the command \shorthandon sets the \catcode to active (13). Both commands only work on 'known' shorthand characters. If a character is not known to be a shorthand character its category code will be left unchanged.

**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.

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

The command `\usesshorthands` initiates the definition of user-defined shorthand sequences. It has one argument, the character that starts these personal shorthands.

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

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

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

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

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

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

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

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

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

However, behaviour 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 could then set:

```
\defineshorthand[*polish,*portugese]{"-"}{\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.

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

The command `\aliasshorthand` can be used to let another character perform the same functions as the default shorthand character. If one prefers for example to use the character / over " in typing Polish texts, this can be achieved by entering `\aliasshorthand{/}{/}`.

**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.

`\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>6</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 with, for example, `\usesshorthands`.)

Very often, this is a more convenient way to deactivate shorthands than `\shorthandoff`, as 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 `\bb1@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.)

For your records, here is a list of shorthands, but you must double check them, as they may change.<sup>7</sup>

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

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

**Basque** " ' ~

**Breton** : ; ? !

**Catalan** " ' `

**Czech** " -

<sup>6</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>7</sup>Thanks to Enrico Gregorio

**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>8</sup>

### 1.11 Package options

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

- KeepShorthandsActive** Tells babel not to deactivate shorthands after loading a language file, so that they are also available in the preamble.
- activeacute** For some languages babel supports this options to set ' as a shorthand in case it is not done by default.
- activegrave** Same for `.
- shorthands=**  $\langle char \rangle \langle char \rangle \dots$  | off  
The only language shorthands activated are those given, like, eg:

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

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

- safe=** none | ref | bib  
Some L<sup>A</sup>T<sub>E</sub>X macros are redefined so that using shorthands is safe. With safe=bib only \nocite, \bibcite and \bibitem are redefined. With safe=ref only \newlabel, \ref and \pageref are redefined (as well as a few macros from varioref and ifthen). With safe=none no macro is redefined. This option is strongly recommended, because a good deal of incompatibilities and errors are related to these redefinitions – of course, in such a case you cannot use shorthands in these macros, but this is not a real problem (just use “allowed” characters).

<sup>8</sup>This declaration serves to nothing, but it is preserved for backward compatibility.

- math=** active | normal
- Shorthands are mainly intended for text, not for math. By setting this option with the value `normal` they are deactivated in math mode (default is `active`) and things like  $\{a'\}$  (a closing brace after a shorthand) are not a source of trouble any more.
- config=**  $\langle file \rangle$
- Load  $\langle file \rangle$ .`cfg` instead of the default config file `bblopts.cfg` (the file is loaded even with `noconfigs`).
- main=**  $\langle language \rangle$
- Sets the main language, as explained above, ie, this language is always loaded last. If it is not given as package or global option, it is added to the list of requested languages.
- headfoot=**  $\langle language \rangle$
- By default, headlines and footlines are not touched (only marks), and if they contain language dependent macros (which is not usual) there may be unexpected results. With this option you may set the language in heads and foots.
- noconfigs** Global and language default config files are not loaded, so you can make sure your document is not spoiled by an unexpected `.cfg` file. However, if the key `config` is set, this file is loaded.
- showlanguages** Prints to the log the list of languages loaded when the format was created: number (remember dialects can share it), name, hyphenation file and exceptions file.
- nocase** **New 3.9l** Language settings for uppercase and lowercase mapping (as set by `\SetCase`) are ignored. Use only if there are incompatibilities with other packages.
- silent** **New 3.9l** No warnings and no *infos* are written to the log file.<sup>9</sup>
- strings=** generic | unicode | encoded |  $\langle label \rangle$  |  $\langle font encoding \rangle$
- Selects the encoding of strings in languages supporting this feature. Predefined labels are `generic` (for traditional  $\TeX$ , LICR and ASCII strings), `unicode` (for engines like `xetex` and `luatex`) and `encoded` (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in `\MakeUppercase` and the like (this feature misuses some internal  $\LaTeX$  tools, so use it only as a last resort).
- hyphenmap=** off | main | select | other | other\*
- New 3.9g** Sets the behaviour of case mapping for hyphenation, provided the language defines it.<sup>10</sup> It can take the following values:
- off** deactivates this feature and no case mapping is applied;
- first** sets it at the first switching commands in the current or parent scope (typically, when the aux file is first read and at `\begin{document}`), but also the first `\selectlanguage` in the preamble), and it's the default if a single language option has been stated;<sup>11</sup>

<sup>9</sup>You can use alternatively the package `silence`.

<sup>10</sup>Turned off in plain.

<sup>11</sup>Duplicated options count as several ones.

`select` sets it only at `\selectlanguage`;  
`other` also sets it at `otherlanguage`;  
`other*` also sets it at `otherlanguage*` as well as in heads and foots (if the option `headfoot` is used) and in auxiliary files (ie, at `\select@language`), and it's the default if several language options have been stated. The option `first` can be regarded as an optimized version of `other*` for monolingual documents.<sup>12</sup>

`bidi=`

**New 3.14** Selects the bidi algorithm to be used in `luatex` and `xetex`. See sec. 1.20.

`layout=`

**New 3.16** Selects which layout elements are adapted in bidi documents. See sec. 1.20.

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

## 1.13 ini files

An alternative approach to define a language is by means of an `ini` file. Currently `babel` provides about 200 of these files containing the basic data required for a language. 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

<sup>12</sup>Providing `foreign` is pointless, because the case mapping applied is that at the end of paragraph, but if either `xetex` or `luatex` change this behaviour it might be added. On the other hand, `other` is provided even if I [JBL] think it isn't really useful, but who knows.

compatibility is important). The following section shows how to make use of them currently (by means of `\babelprovide`), but a higher interface, based on package options, is under development.

**EXAMPLE** Although Georgian has its own `ldf` file, here is how to declare this language with an `ini` file in Unicode engines. The `nil` language is required, because currently `babel` raises an error if there is no language.

```

\documentclass{book}

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

\babelfont{rm}{DejaVu Sans}

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}

```

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

---

af	Afrikaans <sup>u</sup>	chr	Cherokee
agq	Aghem	ckb	Central Kurdish
ak	Akan	cs	Czech <sup>u</sup>
am	Amharic <sup>u</sup>	cy	Welsh <sup>u</sup>
ar	Arabic <sup>u</sup>	da	Danish <sup>u</sup>
as	Assamese	dav	Taita
asa	Asu	de-AT	German <sup>u</sup>
ast	Asturian <sup>u</sup>	de-CH	German <sup>u</sup>
az-Cyrl	Azerbaijani	de	German <sup>u</sup>
az-Latn	Azerbaijani	dje	Zarma
az	Azerbaijani <sup>u</sup>	dsb	Lower Sorbian <sup>u</sup>
bas	Basaa	dua	Duala
be	Belarusian <sup>u</sup>	dyo	Jola-Fonyi
bem	Bemba	dz	Dzongkha
bez	Bena	ebu	Embu
bg	Bulgarian <sup>u</sup>	ee	Ewe
bm	Bambara	el	Greek <sup>u</sup>
bn	Bangla <sup>u</sup>	en-AU	English <sup>u</sup>
bo	Tibetan <sup>u</sup>	en-CA	English <sup>u</sup>
brx	Bodo	en-GB	English <sup>u</sup>
bs-Cyrl	Bosnian	en-NZ	English <sup>u</sup>
bs-Latn	Bosnian <sup>u</sup>	en-US	English <sup>u</sup>
bs	Bosnian <sup>u</sup>	en	English <sup>u</sup>
ca	Catalan <sup>u</sup>	eo	Esperanto <sup>u</sup>
ce	Chechen	es-MX	Spanish <sup>u</sup>
cgg	Chiga	es	Spanish <sup>u</sup>

et	Estonian <sup>ul</sup>	ko	Korean
eu	Basque <sup>ul</sup>	kok	Konkani
ewo	Ewondo	ks	Kashmiri
fa	Persian <sup>ul</sup>	ksb	Shambala
ff	Fulah	ksf	Bafia
fi	Finnish <sup>ul</sup>	ksh	Colognian
fil	Filipino	kw	Cornish
fo	Faroese	ky	Kyrgyz
fr	French <sup>ul</sup>	lag	Langi
fr-BE	French <sup>ul</sup>	lb	Luxembourgish
fr-CA	French <sup>ul</sup>	lg	Ganda
fr-CH	French <sup>ul</sup>	lkt	Lakota
fr-LU	French <sup>ul</sup>	ln	Lingala
fur	Friulian <sup>ul</sup>	lo	Lao <sup>ul</sup>
fy	Western Frisian	lrc	Northern Luri
ga	Irish <sup>ul</sup>	lt	Lithuanian <sup>ul</sup>
gd	Scottish Gaelic <sup>ul</sup>	lu	Luba-Katanga
gl	Galician <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>l</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>l</sup>
hy	Armenian	ms-SG	Malay <sup>l</sup>
ia	Interlingua <sup>ul</sup>	ms	Malay <sup>ul</sup>
id	Indonesian <sup>ul</sup>	mt	Maltese
ig	Igbo	mua	Mundang
ii	Sichuan Yi	my	Burmese
is	Icelandic <sup>ul</sup>	mzn	Mazanderani
it	Italian <sup>ul</sup>	naq	Nama
ja	Japanese	nb	Norwegian Bokmål <sup>ul</sup>
jgo	Ngomba	nd	North Ndebele
jmc	Machame	ne	Nepali
ka	Georgian <sup>ul</sup>	nl	Dutch <sup>ul</sup>
kab	Kabyle	nmg	Kwasio
kam	Kamba	nn	Norwegian Nynorsk <sup>ul</sup>
kde	Makonde	nnh	Ngiemboon
kea	Kabuverdianu	nus	Nuer
khq	Koyra Chiini	nyn	Nyankole
ki	Kikuyu	om	Oromo
kk	Kazakh	or	Odia
kkj	Kako	os	Ossetic
kl	Kalaallisut	pa-Arab	Punjabi
kln	Kalenjin	pa-Guru	Punjabi
km	Khmer	pa	Punjabi
kn	Kannada <sup>ul</sup>	pl	Polish <sup>ul</sup>

pms	Piedmontese <sup>ul</sup>	sw	Swahili
ps	Pashto	ta	Tamil <sup>u</sup>
pt-BR	Portuguese <sup>ul</sup>	te	Telugu <sup>ul</sup>
pt-PT	Portuguese <sup>ul</sup>	teo	Teso
pt	Portuguese <sup>ul</sup>	th	Thai <sup>ul</sup>
qu	Quechua	ti	Tigrinya
rm	Romansh <sup>ul</sup>	tk	Turkmen <sup>ul</sup>
rn	Rundi	to	Tongan
ro	Romanian <sup>ul</sup>	tr	Turkish <sup>ul</sup>
rof	Rombo	twq	Tasawaq
ru	Russian <sup>ul</sup>	tzm	Central Atlas Tamazight
rw	Kinyarwanda	ug	Uyghur
rwk	Rwa	uk	Ukrainian <sup>ul</sup>
sah	Sakha	ur	Urdu <sup>ul</sup>
saq	Samburu	uz-Arab	Uzbek
sbp	Sangu	uz-Cyrl	Uzbek
se	Northern Sami <sup>ul</sup>	uz-Latn	Uzbek
seh	Sena	uz	Uzbek
ses	Koyraboro Senni	vai-Latn	Vai
sg	Sango	vai-Vaii	Vai
shi-Latn	Tachelhit	vai	Vai
shi-Tfng	Tachelhit	vi	Vietnamese <sup>ul</sup>
shi	Tachelhit	vun	Vunjo
si	Sinhala	wae	Walser
sk	Slovak <sup>ul</sup>	xog	Soga
sl	Slovenian <sup>ul</sup>	yav	Yangben
smn	Inari Sami	yi	Yiddish
sn	Shona	yo	Yoruba
so	Somali	yue	Cantonese
sq	Albanian <sup>ul</sup>	zgh	Standard Moroccan Tamazight
sr-Cyrl-BA	Serbian <sup>ul</sup>	zh-Hans-HK	Chinese
sr-Cyrl-ME	Serbian <sup>ul</sup>	zh-Hans-MO	Chinese
sr-Cyrl-XK	Serbian <sup>ul</sup>	zh-Hans-SG	Chinese
sr-Cyrl	Serbian <sup>ul</sup>	zh-Hans	Chinese
sr-Latn-BA	Serbian <sup>ul</sup>	zh-Hant-HK	Chinese
sr-Latn-ME	Serbian <sup>ul</sup>	zh-Hant-MO	Chinese
sr-Latn-XK	Serbian <sup>ul</sup>	zh-Hant	Chinese
sr-Latn	Serbian <sup>ul</sup>	zh	Chinese
sr	Serbian <sup>ul</sup>	zu	Zulu
sv	Swedish <sup>ul</sup>		

---

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

---

aghem	armenian
akan	assamese
albanian	asturian
american	asu
amharic	australian
arabic	austrian

azerbaijani-cyrillic  
azerbaijani-cyrl  
azerbaijani-latin  
azerbaijani-latn  
azerbaijani  
bafia  
bambara  
basaa  
basque  
belarusian  
bemba  
bena  
bengali  
bodo  
bosnian-cyrillic  
bosnian-cyrl  
bosnian-latin  
bosnian-latn  
bosnian  
brazilian  
breton  
british  
bulgarian  
burmese  
canadian  
cantonese  
catalan  
centralatlastamazight  
centralkurdish  
chechen  
cherokee  
chiga  
chinese-hans-hk  
chinese-hans-mo  
chinese-hans-sg  
chinese-hans  
chinese-hant-hk  
chinese-hant-mo  
chinese-hant  
chinese-simplified-hongkongsarchina  
chinese-simplified-macausarchina  
chinese-simplified-singapore  
chinese-simplified  
chinese-traditional-hongkongsarchina  
chinese-traditional-macausarchina  
chinese-traditional  
chinese  
cognian  
cornish  
croatian  
czech  
danish  
duala  
dutch  
dzongkha  
embu  
english-au  
english-australia  
english-ca  
english-canada  
english-gb  
english-newzealand  
english-nz  
english-unitedkingdom  
english-unitedstates  
english-us  
english  
esperanto  
estonian  
ewe  
ewondo  
faroese  
filipino  
finnish  
french-be  
french-belgium  
french-ca  
french-canada  
french-ch  
french-lu  
french-luxembourg  
french-switzerland  
french  
friulian  
fulah  
galician  
ganda  
georgian  
german-at  
german-austria  
german-ch  
german-switzerland  
german  
greek  
gujarati  
gusii  
hausa-gh  
hausa-ghana  
hausa-ne  
hausa-niger  
hausa  
hawaiian  
hebrew  
hindi  
hungarian  
icelandic  
igbo  
inarisami

indonesian  
interlingua  
irish  
italian  
japanese  
jolafonyi  
kabuverdianu  
kabyle  
kako  
kalaallisut  
kalenjin  
kamba  
kannada  
kashmiri  
kazakh  
khmer  
kikuyu  
kinyarwanda  
konkani  
korean  
koyraborosenni  
koyrachiini  
kwasio  
kyrgyz  
lakota  
langi  
lao  
latvian  
lingala  
lithuanian  
lowersorbian  
lsorbian  
lubakatanga  
luo  
luxembourgish  
luyia  
macedonian  
machame  
makhuwameetto  
makonde  
malagasy  
malay-bn  
malay-brunei  
malay-sg  
malay-singapore  
malay  
malayalam  
maltese  
manx  
marathi  
masai  
mazanderani  
meru  
meta  
mexican  
mongolian  
morisyen  
mundang  
nama  
nepali  
newzealand  
ngiemboon  
ngomba  
norsk  
northernluri  
northernsami  
northndebele  
norwegianbokmal  
norwegiannynorsk  
nswissgerman  
nuer  
nyankole  
nynorsk  
occitan  
oriya  
oromo  
ossetic  
pashto  
persian  
piedmontese  
polish  
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  
scottishgaelic  
sena  
serbian-cyrillic-bosniaherzegovina  
serbian-cyrillic-kosovo  
serbian-cyrillic-montenegro

serbian-cyrillic	telugu
serbian-cyrl-ba	teso
serbian-cyrl-me	thai
serbian-cyrl-xk	tibetan
serbian-cyrl	tigrinya
serbian-latin-bosniaherzegovina	tongan
serbian-latin-kosovo	turkish
serbian-latin-montenegro	turkmen
serbian-latin	ukenglish
serbian-latn-ba	ukrainian
serbian-latn-me	upporsorbian
serbian-latn-xk	urdu
serbian-latn	usenglish
serbian	usorbian
shambala	uyghur
shona	uzbek-arab
sichuanyi	uzbek-arabic
sinhala	uzbek-cyrillic
slovak	uzbek-cyrl
slovene	uzbek-latin
slovenian	uzbek-latn
soga	uzbek
somali	vai-latin
spanish-mexico	vai-latn
spanish-mx	vai-vai
spanish	vai-vaii
standardmoroccantamazight	vai
swahili	vietnam
swedish	vietnamese
swissgerman	vunjo
tachelhit-latin	walser
tachelhit-latn	welsh
tachelhit-tfng	westernfrisian
tachelhit-tifinagh	yangben
tachelhit	yiddish
taita	yoruba
tamil	zarma
tasawaq	zulu afrikaans

---

## 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*}

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. Alternatively, you may set a font for a script – just precede its name (lowercase) with a star (eg, `*devanagari`).

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

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

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

```
\documentclass{article}

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

\babelprovide[import=he]{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 could replace the red line above with, say:

```
\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 ones.

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

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

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

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

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

This makes sure the OpenType script for Devanagari is `deva` and not `dev2`.

**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 (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 are 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 could be problematic, and also 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 behaviour.

**WARNING** Do not use `\setxxxxfont` and `\babelfont` at the same time. `\babelfont` follows the standard  $\text{\TeX}$  conventions to set the basic families – define `\xxdefault`, and activate it with `\xxfamily`. On the other hand, `\setxxxxfont` in `fontspec` takes a different approach, because `\xxfamily` is redefined with the family name hardcoded (so that `\xxdefault` becomes `no-op`). Of course, both methods are incompatible, and if you use `\setxxxxfont`, font switching with `\babelfont` just does *not* work (nor the standard `\xxdefault`, for that matter).

## 1.15 Modifying a language

Modifying the behaviour of a language (say, the chapter “caption”), is sometimes necessary, but not always trivial.

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

```
\addto\captionenglish{%  
  \renewcommand\contentsname{Foo}%  
}
```

As of 3.15, there is no need to hide spaces with `%` (`babel` removes them), but it is advisable to do it.

- 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}
```

- Macros to be run when a language is selected can be add to `\extras<lang>`:

```
\addto\extrarussian{\mymacro}
```

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

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

## 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.

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

Defines the internal structure of the language with some defaults: the hyphen rules, if not available, are set to the current ones, left and right hyphen mins are set to 2 and 3, but captions and date are not defined. Conveniently, babel warns you about what to do. Very likely you will find alerts like that in the log file:

```
Package babel Warning: \mylangchaptername not set. Please, define
(babel)                it in the preamble with something like:
(babel)                \renewcommand\mylangchaptername{..}
(babel)                Reported on input line 18.
```

In most cases, you will only need to define a few macros.

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

```
\usepackage[danish]{babel}
\babelprovide{arhinish}
\renewcommand\arhinishchaptername{Chapitula}
\renewcommand\arhinishrefname{Refirenke}
\renewcommand\arhinishhyphenmins{22}
```

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, date, and hyphenmins. 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.

There are about 200 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 will show a warning about the current lack of suitability of the date format (hindi, french, breton, and occitan).

Besides `\today`, there is a `\<language>date` macro with three arguments: year, month and day numbers. In fact, `\today` calls `\<language>today`, which in turn calls `\<language>date{\the\year}{\the\month}{\the\day}`.

`captions=` *language-tag*

Loads only the strings. For example:

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

**hyphenrules=**  $\langle$ *language-list* $\rangle$

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

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

If none of the listed hyphenrules exist, the default behaviour 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).

**main** This valueless option makes the language the main one. Only in newly defined languages.

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

**New 3.15** Sets the script name to be used by fontspec (eg, Devanagar i). Overrides the value in the ini file. This value is particularly important because it sets the writing direction.

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

**New 3.15** Sets the language name to be used by fontspec (eg, Hindi). Overrides the value in the ini file. Not so important, but sometimes still relevant.

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

## 1.17 Getting the current language name

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

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

**\iflanguage**  $\langle$ *language* $\rangle$  $\langle$ *true* $\rangle$  $\langle$ *false* $\rangle$

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

**WARNING** The advice about `\languagename` also applies here – use `iflang` instead of `\iflanguage` if possible.

## 1.18 Hyphenation tools

`\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 behaviour 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 enabling 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 `\babelnu1lhyphen`; (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.

`\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.

## 1.19 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>15</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>16</sup>

`\ensureascii` {*<text>*}

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

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

The foregoing rules (which are applied “at begin document”) cover most of 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.

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

<sup>15</sup>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>16</sup>But still defined for backwards compatibility.

## 1.20 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 could 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** Setting bidi text has many subtleties (see for example <https://www.w3.org/TR/html-bidi/>). This means the babel bidi code may take some time before it is truly stable. 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).

There are some package options controlling bidi writing.

**bidi=** default | basic-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.

**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, which will be improved in the future. Remember basic-r is available in luatex only.<sup>17</sup>

```
\documentclass{article}

\usepackage[nil, bidi=basic-r]{babel}

\babelprovide[import=ar, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

    وقد عرفت شبه جزيرة العرب طيلة العصر الهيليني (الاعريقي) بـ
    Arabia أو Aravia (بالاعريقية 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}
```

**layout=** sectioning | counters | lists | contents | columns

**New 3.16** Selects which layout elements are adapted in bidi documents. This list will be expanded in future releases (footnotes, tables, etc.). *To be expanded*. Note some options are not required in luatex.

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

<sup>17</sup>At the time of this writing some Arabic fonts are not rendered correctly by the default luatex font loader, with misplaced kerns inside some words, so double check the resulting text. It seems a fix is on the way.

**counters** required in all engines to reorder correctly section numbers and the like (eg,  $\langle subsection \rangle.\langle section \rangle$ ); required in xetex and pdftex for counters in general, as well as in luatex with `bidi=default`;

**lists** required in xetex and pdftex, but only in multilingual documents in luatex;

**contents** required in xetex and pdftex; in luatex toc entries are R by default;

**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 (including multicol).

`\babelsublr`  $\langle lr-text \rangle$

However, digits in pdftex must be marked up explicitly (unlike luatex with `bidi=basic-r` and, usually, xetex). Mainly for it (although available in all engines, because it can be useful), this command is provided to set  $\langle lr-text \rangle$  in L mode. It's mainly intended for what Unicode calls weak characters, because words are best set with the corresponding language. For this reason, there is no `rl` counterpart.

`\BabelPatchSection`  $\langle section-name \rangle$

Mainly for bidi text, but it could 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, but with this command you can set them individually if necessary (but note then tocs and marks are not touched).

## 1.21 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).

## 1.22 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`  $\langle name \rangle\langle event \rangle\langle code \rangle$

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

Names containing the string `babel` are reserved (they are used, for example, by `\useshortands*` to add a hook for the event `afterextras`).

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

**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 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 loads `switch.def`. It can be used to load a different version of this files or to load nothing.

**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).

### 1.23 Languages supported by babel

In the following table most of the languages supported by `babel` 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** bahasa, indonesian, indon, bahasai  
**Interlingua** interlingua  
**Irish Gaelic** irish  
**Italian** italian  
**Latin** latin  
**Lower Sorbian** lowersorbian  
**Malay** bahasam, malay, melayu  
**North Sami** samin  
**Norwegian** norsk, nynorsk  
**Polish** polish  
**Portuguese** portuges, portuguese, brazilian, brazil  
**Romanian** romanian  
**Russian** russian  
**Scottish Gaelic** scottish  
**Spanish** spanish  
**Slovakian** slovak  
**Slovenian** slovene  
**Swedish** swedish  
**Serbian** serbian  
**Turkish** turkish  
**Ukrainian** ukrainian  
**Upper Sorbian** upporsorbian  
**Welsh** welsh

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

Most of them work out of the box, but some may require extra fonts, encoding files, a preprocessor or even a complete framework (like CJK). 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 <file>`, which creates `<file>.tex`; you can then typeset the latter with  $\LaTeX$ .

## 1.24 Tips, workarounds, know issues and notes

- If you use the document class `book` and you use `\ref` inside the argument of `\chapter` (or just use `\ref` inside `\MakeUppercase`),  $\LaTeX$  will keep complaining about an undefined label. To prevent such problems, you could 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}}
```

(A recent version of `inputenc` is required.)

- For the hyphenation to work correctly, `lccodes` cannot change, because  $\TeX$  only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.<sup>18</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  $\TeX$ , not of `babel`. Alternatively, you may use `\usesshorthands` to activate `'` and `\defineshorthand`, 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  $\TeX$  enter in an infinite loop in some rare cases. (Another issue in the 'to do' list, although there is a partial solution.)

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

<sup>18</sup>This explains why  $\LaTeX$  assumes the lowercase mapping of T1 and does not provide a tool for multiple mappings. Unfortunately, `\savingshyphcodes` is not a solution either, because `lccodes` for hyphenation are frozen in the format and cannot be changed.

**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.

## 1.25 Current and future work

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).

It is possible now to typeset Arabic or Hebrew with numbers and L text. Next on the roadmap are line breaking in Thai and the like, as well as “non-European” digits. Also on the roadmap are R layouts (lists, footnotes, tables, column order), page and section numbering, and maybe kashida justification.

As to Thai line breaking, here is the basic idea of what luatex can do for us, with the Thai patterns and a little script (the final version will not be so little, of course). It replaces each discretionary by the equivalent to ZWJ.

```

\documentclass{article}

\usepackage[nil]{babel}

\babelprovide[import=th, main]{thai}

\babelfont{rm}{FreeSerif}

\directlua{
local GLYPH = node.id'glyph'
function insertsp (head)
  local size = 0
  for item in node.traverse(head) do
    local i = item.id
    if i == GLYPH then
      f = font.getfont(item.font)
      size = f.size
    elseif i == 7 then
      local n = node.new(12, 0)
      node.setglue(n, 0, size * 1) % 1 is a factor
      node.insert_before(head, item, n)
      node.remove(head, item)
    end
  end
end
end

luatexbase.add_to_callback('hyphenate',

```

```

function (head, tail)
  lang.hyphenate(head)
  insertsp(head)
end, 'insertsp')
}

\begin{document}

(Thai text.)

\end{document}

```

Useful additions would be, for example, time, currency, addresses and personal names.<sup>19</sup> But that is the easy part, because they don't require modifying the  $\LaTeX$  internals. 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.

## 1.26 Tentative and experimental code

Handling of “**Unicode**” fonts is problematic. There is `fontspec`, but special macros are required (not only the NFSS ones) and it doesn't provide “orthogonal axis” for features, including those related to the language (mainly language and script). A couple of tentative macros, were provided by `babel` ( $\geq 3.9g$ ) with a partial solution. These macros are now deprecated – use `\babelfont`.

- `\babelFSstore{<babel-language>}` sets the current three basic families (rm, sf, tt) as the default for the language given.
- `\babelFSdefault{<babel-language>}{<fontspec-features>}` patches `\fontspec` so that the given features are always passed as the optional argument or added to it (not an ideal solution).

So, for example:

```

\setmainfont[Language=Turkish]{Minion Pro}
\babelFSstore{turkish}
\setmainfont{Minion Pro}
\babelFSfeatures{turkish}{Language=Turkish}

```

**Bidi writing** is taking its *first steps*. *First steps* means exactly that. For example, in `luatex` any Arabic text must be marked up explicitly in L mode. On the other hand, `xetex` poses quite different challenges. Document layout (lists, footnotes, etc.) is not touched at all. See the code section for `\foreignlanguage*` (a new starred version of `\foreignlanguage`). `xetex` relies on the font to properly handle these unmarked changes, so it is not under the control of  $\TeX$ .

## 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 $\LaTeX$` ,

<sup>19</sup>See for example POSIX, ISO 14652 and the Unicode Common Locale Data Repository (CLDR). Those system, however, have limited application to  $\TeX$  because their aim is just to display information and not fine typesetting.

pdfL<sup>A</sup>T<sub>E</sub>X). babel provides a tool which has become standard in many distributions and based on a “configuration file” named `language.dat`. The exact way this file is used depends on the distribution, so please, read the documentation for the latter (note also some distributions generate the file with some tool).

**New 3.9q** With luatex, however, patterns are loaded on the fly when requested by the language (except the “0th” language, typically english, which is preloaded always).<sup>20</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>21</sup>

## 2.1 Format

In that file the person who maintains a T<sub>E</sub>X environment has to record for which languages he has hyphenation patterns *and* in which files these are stored<sup>22</sup>. When hyphenation exceptions are stored in a separate file this can be indicated by naming that file *after* the file with the hyphenation patterns.

The file can contain empty lines and comments, as well as lines which start with an equals (=) sign. Such a line will instruct L<sup>A</sup>T<sub>E</sub>X that the hyphenation patterns just processed have to be known under an alternative name. Here is an example:

```
% File      : language.dat
% Purpose   : tell iniTeX what files with patterns to load.
english    english.hyphenations
=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>23</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 could 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.

<sup>20</sup>This feature was added to 3.9o, but it was buggy. Both 3.9o and 3.9p are deprecated.

<sup>21</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>22</sup>This is because different operating systems sometimes use *very* different file-naming conventions.

<sup>23</sup>This is not a new feature, but in former versions it didn't work correctly.

### 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  $\TeX$  users, so the files have to be coded so that they can be read by both  $\LaTeX$  and plain  $\TeX$ . The current format can be checked by looking at the value of the macro `\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 `\langle lang \rangle hyphenmins`, `\langle lang \rangle captions`, `\langle lang \rangle date`, `\langle lang \rangle extras` and `\langle lang \rangle noextras` (the last two may be left empty); where `\langle lang \rangle` is either the name of the language definition file or the name of the  $\LaTeX$  option that is to be used. These macros and their functions are discussed below. You must define all or none for a language (or a dialect); defining, say, `\langle lang \rangle date` but not `\langle lang \rangle captions` does not raise an error but can lead to unexpected results.
- When a language definition file is loaded, it can define `\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  $\LaTeX$  (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>24</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

Now language files are “outsourced” and are located in a separate directory (`/macros/latex/contrib/babel-contrib`), so that they are contributed directly to CTAN (please, do not send to me language styles just to upload them to CTAN). Of course, placing your style files in this directory is not mandatory, but if you want to do it, here are a few guidelines.

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

The following page provides a starting point: <http://www.texnia.com/incubator.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. For older versions of `plain.tex` and `lplain.tex` a substitute definition is used. 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 behaviour 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:

<sup>24</sup>But not removed, for backward compatibility.

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

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

<code>\providehyphenmins</code>	The macro <code>\providehyphenmins</code> should be used in the language definition files to set <code>\lefthyphenmin</code> and <code>\righthyphenmin</code> . 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 <i>not</i> set them).
<code>\captions&lt;lang&gt;</code>	The macro <code>\captions&lt;lang&gt;</code> defines the macros that hold the texts to replace the original hard-wired texts.
<code>\date&lt;lang&gt;</code>	The macro <code>\date&lt;lang&gt;</code> defines <code>\today</code> .
<code>\extras&lt;lang&gt;</code>	The macro <code>\extras&lt;lang&gt;</code> 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.
<code>\noextras&lt;lang&gt;</code>	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 <code>\extras&lt;lang&gt;</code> , a macro that brings $\TeX$ into a predefined state is needed. It will be no surprise that the name of this macro is <code>\noextras&lt;lang&gt;</code> .
<code>\bbl@declare@tribute</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 in sec. 3.8 (babel 3.9 and later).

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

```

\lfdInit{<language>}{captions<language>}

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

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

\bb1@declare@ttribute{<language>}{<attrib>}{%
  \expandafter\addto\expandafter\extras<language>
  \expandafter{\extras<attrib><language>}%
  \let\captions<language>\captions<attrib><language>}

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

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

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

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

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

\EndBabelCommands

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

\lfd@finish{<language>}

```

### 3.4 Support for active characters

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

`\initiate@active@char`

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

`\bbl@activate`  
`\bbl@deactivate`

The command `\bbl@activate` is used to change the way an active character expands. `\bbl@activate` ‘switches on’ the active behaviour 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  
`\bbl@remove@special` macro, representing the set of all characters that have a special category code.” [2, p. 380]  
 It is used to set text ‘verbatim’. To make this work if more characters get a special category  
 code, you have to add this character to the macro `\dospecial`.  $\LaTeX$  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  $\langle char \rangle$  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>25</sup>.

`\babel@save` To save the current meaning of any control sequence, the macro `\babel@save` is provided.  
 It takes one argument,  $\langle csname \rangle$ , 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  $\langle variable \rangle$ .  
 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 could be  
 either global (usually) or local (sometimes). That does not seem very consistent, but this  
 behaviour 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 behaviour of `\bbl@allowhyphens`.

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

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

`\bbl@frenchspacing` The commands `\bbl@frenchspacing` and `\bbl@nonfrenchspacing` can be used to  
`\bbl@nonfrenchspacing` properly switch French spacing on and off.

<sup>25</sup>This mechanism was introduced by Bernd Raichle.

### 3.8 Encoding-dependent strings

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

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

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

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

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

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

If a string is set several times (because several blocks are read), the first one takes precedence (ie, it works much like `\providecommand`).

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

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

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

The  $\langle \textit{category} \rangle$  is either `captions`, `date` or `extras`. You must stick to these three categories, even if no error is raised when using other name.<sup>26</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}
```

<sup>26</sup>In future releases further categories may be added.

```

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

\EndBabelCommands

```

A real example is:

```

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

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

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

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

\StartBabelCommands{german,austrian}{date}
\SetString\monthiiiname{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>27</sup>

<sup>27</sup>This replaces in 3.9g a short-lived  $\backslash UseStrings$  which has been removed because it did not work.

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

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

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

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

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

#1 is replaced by the roman numeral.

`\SetCase`  $[\langle map-list \rangle] \langle toupper-code \rangle \langle tolower-code \rangle$   
Sets globally code to be executed at `\MakeUpperCase` and `\MakeLowerCase`. The code would be typically things like `\let\BB\bb` and `\uccode` or `\lccode` (although for the reasons explained above, changes in lc/uc codes may not work). A  $\langle map-list \rangle$  is a series of macros using the internal format of `@uc1clist` (eg, `\bb\BB\cc\CC`). The mandatory arguments take precedence over the optional one. This command, unlike `\SetString`, is executed always (even without strings), and it is intended for minor readjustments only. For example, as T1 is the default case mapping in L<sup>A</sup>T<sub>E</sub>X, we could set for Turkish:

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

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

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

`\SetHyphenMap` `{\to-lower-macros}`

**New 3.9g** Case mapping serves in T<sub>E</sub>X for two unrelated purposes: case transforms (upper/lower) and hyphenation. `\SetCase` handles the former, while hyphenation is handled by `\SetHyphenMap` and controlled with the package option `hyphenmap`. So, even if internally they are based on the same T<sub>E</sub>X primitive (`\lccode`), `babel` sets them separately. There are three helper macros to be used inside `\SetHyphenMap`:

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

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

```
\SetHyphenMap{\BabelLowerMM{"100}{\lccode{"11F}{2}{\lccode{"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 changes in version 3.9 are 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 behaviour 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 could 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.

## 4.2 Changes in babel version 3.7

In `babel` version 3.7 a number of bugs that were found in version 3.6 are fixed. Also a number of changes and additions have occurred:

- Shorthands are expandable again. The disadvantage is that one has to type `'{ }a` when the acute accent is used as a shorthand character. The advantage is that a number of other problems (such as the breaking of ligatures, etc.) have vanished.
- Two new commands, `\shorthandon` and `\shorthandoff` have been introduced to enable to temporarily switch off one or more shorthands.
- Support for typesetting Hebrew (and potential support for typesetting other right-to-left written languages) is now available thanks to Rama Porrat and Boris Lavva.
- A language attribute has been added to the `\mark . . .` commands in order to make sure that a Greek header line comes out right on the last page before a language switch.
- Hyphenation pattern files are now read *inside a group*; therefore any changes a pattern file needs to make to lowercase codes, uppercase codes, and category codes are kept local to that group. If they are needed for the language, these changes will need to be repeated and stored in `\extras . . .`
- The concept of language attributes is introduced. It is intended to give the user some control over the features a language-definition file provides. Its first use is for the Greek language, where the user can choose the `πολυτονικό` (“polytonikó” or multi-accented) Greek way of typesetting texts.
- The environment `hyphenrules` is introduced.
- The syntax of the file `language.dat` has been extended to allow (optionally) specifying the font encoding to be used while processing the patterns file.
- The command `\providehyphenmins` should now be used in language definition files in order to be able to keep any settings provided by the pattern file.

## Part II

# The 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 `babel` package after unpacking consists of the following files:

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

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

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

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

**hyphen.cfg** is the file to be used when generating the formats to load hyphenation patterns. By default it also loads `switch.def`.

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.

```
1 <<version=3.16>>
2 <<date=2018/01/04>>
```

## 6 Tools

**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 behaviour of the latter. Used in `babel.def` and in `babel.sty`, which means in  $\LaTeX$  is executed twice, but we need them when defining options and `babel.def` cannot be loaded until options have been defined. This does not hurt, but should be fixed somehow.

```
3 <<*Basic macros>> ≡
4 \def\bbl@stripslash{\expandafter@gobble\string}
5 \def\bbl@add#1#2{%
6   \bbl@ifunset{\bbl@stripslash#1}%
7     {\def#1{#2}}%
8     {\expandafter\def\expandafter#1\expandafter{#1#2}}
9 \def\bbl@xin@{\@expandtwoargs\in@}
10 \def\bbl@csarg#1#2{\expandafter#1\csname bbl@#2\endcsname}%
11 \def\bbl@cs#1{\csname bbl@#1\endcsname}
12 \def\bbl@loop#1#2#3{\bbl@loop#1{#3}#2, \@nnil,}
13 \def\bbl@loopx#1#2{\expandafter\bbl@loop\expandafter#1\expandafter{#2}}
14 \def\bbl@@loop#1#2#3,{%
15   \ifx\@nnil#3\relax\else
16     \def#1{#3}#2\bbl@afterfi\bbl@@loop#1{#2}%
17   \fi}
18 \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.

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

`\bbl@afterelse` Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the `\else` and `\fi` parts of an `\if`-statement<sup>28</sup>. These

`\bbl@afterfi`

<sup>28</sup>This code is based on code presented in TUGboat vol. 12, no2, June 1991 in “An expansion Power Lemma” by Sonja Maus.

macros will break if another `\if... \fi` statement appears in one of the arguments and it is not enclosed in braces.

```
25 \long\def\bbl@afterelse#1\else#2\fi{\fi#1}
26 \long\def\bbl@afterfi#1\fi{\fi#1}
```

`\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.

```
27 \def\bbl@tempa#1{%
28   \long\def\bbl@trim##1##2{%
29     \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
30   \def\bbl@trim@c{%
31     \ifx\bbl@trim@a\@sptoken
32       \expandafter\bbl@trim@b
33     \else
34       \expandafter\bbl@trim@b\expandafter#1%
35     \fi}%
36   \long\def\bbl@trim@b#1##1 \@nil{\bbl@trim@i##1}}
37 \bbl@tempa{ }
38 \long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
39 \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.

```
40 \def\bbl@ifunset#1{%
41   \expandafter\ifx\csname#1\endcsname\relax
42     \expandafter\@firstoftwo
43   \else
44     \expandafter\@secondoftwo
45   \fi}
46 \bbl@ifunset{ifcsname}%
47 {}%
48 {\def\bbl@ifunset#1{%
49   \ifcsname#1\endcsname
50     \expandafter\ifx\csname#1\endcsname\relax
51       \bbl@afterelse\expandafter\@firstoftwo
52     \else
53       \bbl@afterfi\expandafter\@secondoftwo
54     \fi
55   \else
56     \expandafter\@firstoftwo
57   \fi}}
```

`\bbl@ifblank` A tool from `url`, by Donald Arseneau, which tests if a string is empty or space.

```
58 \def\bbl@ifblank#1{%
59   \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
60 \long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
```

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).

```
61 \def\bbl@forkv#1#2{%
```

```

62 \def\bbl@kvcmd##1##2##3{#2}%
63 \bbl@kvnext#1,\@nil,}
64 \def\bbl@kvnext#1,{%
65 \ifx\@nil#1\relax\else
66 \bbl@ifblank{#1}{\bbl@forkv@eq#1=\@empty=\@nil{#1}}%
67 \expandafter\bbl@kvnext
68 \fi}
69 \def\bbl@forkv@eq#1=#2=#3\@nil#4{%
70 \bbl@trim@def\bbl@forkv@a{#1}%
71 \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).

```

72 \def\bbl@vforeach#1#2{%
73 \def\bbl@forcmd##1{#2}%
74 \bbl@fornext#1,\@nil,}
75 \def\bbl@fornext#1,{%
76 \ifx\@nil#1\relax\else
77 \bbl@ifblank{#1}{\bbl@trim\bbl@forcmd{#1}}%
78 \expandafter\bbl@fornext
79 \fi}
80 \def\bbl@foreach#1{\expandafter\bbl@vforeach\expandafter{#1}}

```

`\bbl@replace`

```

81 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
82 \toks@{}}%
83 \def\bbl@replace@aux##1#2##2#2{%
84 \ifx\bbl@nil##2%
85 \toks@\expandafter{\the\toks@##1}%
86 \else
87 \toks@\expandafter{\the\toks@##1#3}%
88 \bbl@afterfi
89 \bbl@replace@aux##2#2%
90 \fi}%
91 \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
92 \edef#1{\the\toks@}}

```

`\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.

```

93 \def\bbl@exp#1{%
94 \begingroup
95 \let\ \noexpand
96 \def\<##1>{\expandafter\noexpand\csname##1\endcsname}%
97 \edef\bbl@exp@aux{\endgroup#1}%
98 \bbl@exp@aux}

```

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 pdfTeX, 1 is luatex, and 2 is xetex. You may use the latter it in your language style if you want.

```

99 \def\bbl@ifsamestring#1#2{%
100 \begingroup
101 \protected@edef\bbl@tempb{#1}%
102 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
103 \protected@edef\bbl@tempc{#2}%
104 \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
105 \ifx\bbl@tempb\bbl@tempc

```

```

106     \aftergroup\@firstoftwo
107     \else
108     \aftergroup\@secondoftwo
109     \fi
110 \endgroup}
111 \chardef\bbl@engine=%
112 \ifx\directlua\@undefined
113     \ifx\XeTeXinputencoding\@undefined
114         \z@
115     \else
116         \tw@
117     \fi
118 \else
119     \@ne
120 \fi
121 <</Basic macros>>

```

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

```

122 << *Make sure ProvidesFile is defined >> ≡
123 \ifx\ProvidesFile\@undefined
124     \def\ProvidesFile#1[#2 #3 #4]{%
125         \wlog{File: #1 #4 #3 <#2>}%
126         \let\ProvidesFile\@undefined}
127 \fi
128 <</Make sure ProvidesFile is defined >>

```

The following code is used in `babel.sty` and `babel.def`, and loads (only once) the data in `language.dat`.

```

129 << *Load patterns in luatex >> ≡
130 \ifx\directlua\@undefined\else
131     \ifx\bbl@luapatterns\@undefined
132         \input luababel.def
133     \fi
134 \fi
135 <</Load patterns in luatex >>

```

The following code is used in `babel.def` and `switch.def`.

```

136 << *Load macros for plain if not LaTeX >> ≡
137 \ifx\AtBeginDocument\@undefined
138     \input plain.def\relax
139 \fi
140 <</Load macros for plain if not LaTeX >>

```

## 6.1 Multiple languages

`\language` Plain  $\TeX$  version 3.0 provides the primitive `\language` that is used to store the current language. When used with a pre-3.0 version this function has to be implemented by allocating a counter. The following block is used in `switch.def` and `hyphen.cfg`; the latter may seem redundant, but remember `babel` doesn't require loading `switch.def` in the format.

```

141 << *Define core switching macros >> ≡
142 \ifx\language\@undefined
143     \csname newcount\endcsname\language
144 \fi
145 <</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` To add languages to T<sub>E</sub>X's memory plain T<sub>E</sub>X version 3.0 supplies `\newlanguage`, in a pre-3.0 environment a similar macro has to be provided. For both cases a new macro is defined here, because the original `\newlanguage` was defined to be `\outer`. For a format based on plain version 2.x, the definition of `\newlanguage` can not be copied because `\count 19` is used for other purposes in these formats. Therefore `\addlanguage` is defined using a definition based on the macros used to define `\newlanguage` in plain T<sub>E</sub>X version 3.0. For formats based on plain version 3.0 the definition of `\newlanguage` can be simply copied, removing `\outer`. Plain T<sub>E</sub>X version 3.0 uses `\count 19` for this purpose.

```

146 <<*Define core switching macros>> ≡
147 \ifx\newlanguage\undefined
148 \csname newcount\endcsname\last@language
149 \def\addlanguage#1{%
150   \global\advance\last@language\@ne
151   \ifnum\last@language<\@cclvi
152     \else
153       \errmessage{No room for a new \string\language!}%
154     \fi
155   \global\chardef#1\last@language
156   \wlog{\string#1 = \string\language\the\last@language}}
157 \else
158   \countdef\last@language=19
159   \def\addlanguage{\alloc@9\language\chardef\@cclvi}
160 \fi
161 <</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 L<sup>A</sup>T<sub>E</sub>X 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 The Package File (L<sup>A</sup>T<sub>E</sub>X, `babel.sty`)

In order to make use of the features of L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>, the `babel` system contains a package file, `babel.sty`. This file is loaded by the `\usepackage` command and defines all the language options whose name is different from that of the `.ldf` file (like variant spellings). It 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 behaviour 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.

## 7.1 base

The first option to be processed is base, which set the hyphenation patterns then resets `ver@babel.sty` so that  $\LaTeX$  forgets about the first loading. After `switch.def` has been loaded (above) and `\AfterBabelLanguage` defined, exits.

```
162 (*package)
163 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
164 \ProvidesPackage{babel}[\langle\date\rangle \langle\version\rangle The Babel package]
165 \@ifpackagewith{babel}{debug}
166   {\let\bbl@debug\@firstofone}
167   {\let\bbl@debug\@gobble}
168 \input switch.def\relax
169 \langle\Load patterns in luatex\rangle
170 \langle\Basic macros\rangle
171 \def\AfterBabelLanguage#1{%
172   \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.

```
173 \ifx\bbl@languages\undefined\else
174   \begingroup
175     \catcode\^^I=12
176     \@ifpackagewith{babel}{showlanguages}{%
177       \begingroup
178         \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
179         \wlog{<*languages>}%
180         \bbl@languages
181         \wlog{</languages>}%
182       \endgroup}{%
183     \endgroup
184     \def\bbl@elt#1#2#3#4{%
185       \ifnum#2=\z@
186         \gdef\bbl@nulllanguage{#1}%
187         \def\bbl@elt##1##2##3##4{%
188           \fi}%
189     \bbl@languages
190 \fi
191 \ifodd\bbl@engine
192   \@ifpackagewith{babel}{bidi=basic-r}{% must go before any \DeclareOption
193     \let\bbl@beforeforeign\leavevmode
194     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}%
195     \RequirePackage{luatexbase}%
196     \directlua{
197       require('babel-bidi.lua')
198       require('babel-bidi-basic-r.lua')
199       luatexbase.add_to_callback('pre_linebreak_filter',
200         Babel.pre_otfload,
201         'Babel.pre_otfload',
202         luatexbase.priority_in_callback('pre_linebreak_filter',
203           'luaotfload.node_processor') or nil)
204       luatexbase.add_to_callback('hpack_filter',
205         Babel.pre_otfload,
206         'Babel.pre_otfload',
207         luatexbase.priority_in_callback('hpack_filter',
208           'luaotfload.node_processor') or nil)}}%
209 \fi
```

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`. Useful for old versions of `polyglossia`, too.

```

210 \@ifpackagewith{babel}{base}{%
211   \ifx\directlua\undefined
212     \DeclareOption*{\bbl@patterns{\CurrentOption}}%
213   \else
214     \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
215   \fi
216   \DeclareOption{base}{}%
217   \DeclareOption{showlanguages}{}%
218   \ProcessOptions
219   \global\expandafter\let\csname opt@babel.sty\endcsname\relax
220   \global\expandafter\let\csname ver@babel.sty\endcsname\relax
221   \global\let@ifl@ter@@\ifl@ter
222   \def@ifl@ter#1#2#3#4#5{\global\let@ifl@ter\ifl@ter@@}%
223   \endinput}{}%

```

## 7.2 key=value options and other general option

The following macros extract language modifiers, and only real package options are kept in the option list. Modifiers are saved and assigned to `\BabelModifiers` at `\bbl@load@language`; when no modifiers have been given, the former is `\relax`. How modifiers are handled are left to language styles; they can use `\in@`, loop them with `\@for` or `load keyval`, for example.

```

224 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
225 \def\bbl@tempb#1.#2{%
226   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
227 \def\bbl@tempd#1.#2@\nnil{%
228   \ifx\@empty#2%
229     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
230   \else
231     \in@{=}{#1}\ifin@
232     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
233   \else
234     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
235     \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
236   \fi
237 \fi}
238 \let\bbl@tempc\@empty
239 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
240 \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.

```

241 \DeclareOption{KeepShorthandsActive}{}
242 \DeclareOption{activeacute}{}
243 \DeclareOption{activegrave}{}
244 \DeclareOption{debug}{}
245 \DeclareOption{noconfigs}{}
246 \DeclareOption{showlanguages}{}
247 \DeclareOption{silent}{}
248 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
249 <<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.

```
250 \let\bbbl@opt@shorthands\@nnil
251 \let\bbbl@opt@config\@nnil
252 \let\bbbl@opt@main\@nnil
253 \let\bbbl@opt@headfoot\@nnil
254 \let\bbbl@opt@layout\@nnil
```

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

```
255 \def\bbbl@tempa#1=#2\bbbl@tempa{%
256   \bbbl@csarg\ifx{opt@#1}\@nnil
257   \bbbl@csarg\edef{opt@#1}{#2}%
258   \else
259     \bbbl@error{%
260       Bad option `#1=#2'. Either you have misspelled the\\%
261       key or there is a previous setting of `#1'}{%
262       Valid keys are `shorthands', `config', `strings', `main',\\%
263       `headfoot', `safe', `math', among others.}
264   \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 \bbbl@language@opts, because they are language options.

```
265 \let\bbbl@language@opts\@empty
266 \DeclareOption*{%
267   \bbbl@xin@{\string=}{\CurrentOption}%
268   \ifin@
269     \expandafter\bbbl@tempa\CurrentOption\bbbl@tempa
270   \else
271     \bbbl@add@list\bbbl@language@opts{\CurrentOption}%
272   \fi}
```

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

```
273 \ProcessOptions*
```

### 7.3 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@ifshorthands is always true, and it is always false if shorthands is empty. Also, some code makes sense only with shorthands=...

```
274 \def\bbbl@sh@string#1{%
275   \ifx#1\@empty\else
276     \ifx#1t\string-%
277     \else\ifx#1c\string,%
278     \else\string#1%
279   \fi\fi
280   \expandafter\bbbl@sh@string
281 \fi}
282 \ifx\bbbl@opt@shorthands\@nnil
283   \def\bbbl@ifshorthand#1#2#3{#2}%
284 \else\ifx\bbbl@opt@shorthands\@empty
285   \def\bbbl@ifshorthand#1#2#3{#3}%
286 \else
```

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

```
287 \def\bbl@ifshorthand#1{%
288   \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
289   \ifin@
290     \expandafter\@firstoftwo
291   \else
292     \expandafter\@secondoftwo
293   \fi}
```

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

```
294 \edef\bbl@opt@shorthands{%
295   \expandafter\bbl@sh@string\bbl@opt@shorthands\@empty}%
```

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

```
296 \bbl@ifshorthand{'}%
297   {\PassOptionsToPackage{activeacute}{babel}}{}
298 \bbl@ifshorthand{`}%
299   {\PassOptionsToPackage{activegrave}{babel}}{}
300 \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.

```
301 \ifx\bbl@opt@headfoot\@nnil\else
302   \g@addto@macro\@resetactivechars{%
303     \set@typeset@protect
304     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
305     \let\protect\noexpand}
306 \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.

```
307 \ifx\bbl@opt@safe\@undefined
308   \def\bbl@opt@safe{BR}
309 \fi
310 \ifx\bbl@opt@main\@nnil\else
311   \edef\bbl@language@opts{%
312     \ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
313     \bbl@opt@main}
314 \fi
```

For layout an auxiliary macro is provided, available for packages and language styles.

```
315 \ifx\bbl@opt@layout\@nnil
316   \newcommand\IfBabelLayout[3]{#3}%
317 \else
318   \newcommand\IfBabelLayout[1]{%
319     \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
320     \ifin@
321       \expandafter\@firstoftwo
322     \else
323       \expandafter\@secondoftwo
324     \fi}
325 \fi
```

## 7.4 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).

```
326 \let\bbl@afterlang\relax
327 \let\BabelModifiers\relax
328 \let\bbl@loaded@empty
329 \def\bbl@load@language#1{%
330   \InputIfFileExists{#1.ldf}%
331   {\edef\bbl@loaded{\CurrentOption
332     \ifx\bbl@loaded@empty\else,\bbl@loaded\fi}%
333     \expandafter\let\expandafter\bbl@afterlang
334       \csname\CurrentOption.ldf-h@k\endcsname
335     \expandafter\let\expandafter\BabelModifiers
336       \csname bbl@mod@\CurrentOption\endcsname}%
337   {\bbl@error{%
338     Unknown option `\'CurrentOption'. Either you misspelled it\\%
339     or the language definition file \CurrentOption.ldf was not found}{%
340     Valid options are: shorthands=, KeepShorthandsActive,\\%
341     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
342     headfoot=, strings=, config=, hyphenmap=, or a language name.}}}
```

Now, we set language options whose names are different from ldf files.

```
343 \def\bbl@try@load@lang#1#2#3{%
344   \IfFileExists{\CurrentOption.ldf}%
345   {\bbl@load@language{\CurrentOption}}%
346   {#1\bbl@load@language{#2}#3}}
347 \DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}{}}
348 \DeclareOption{brazil}{\bbl@try@load@lang{}{portuges}{}}
349 \DeclareOption{brazilian}{\bbl@try@load@lang{}{portuges}{}}
350 \DeclareOption{hebrew}{%
351   \input{rlbabel.def}%
352   \bbl@load@language{hebrew}}
353 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
354 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
355 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
356 \DeclareOption{polutonikogreek}{%
357   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
358 \DeclareOption{portuguese}{\bbl@try@load@lang{}{portuges}{}}
359 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
360 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
361 \DeclareOption{uppersorbian}{\bbl@try@load@lang{}{usorbian}{}}
```

Another way to extend the list of ‘known’ options for babel was to create the file `bblopts.cfg` in which one can add option declarations. However, this mechanism is deprecated – if you want an alternative name for a language, just create a new `.ldf` file loading the actual one. You can also set the name of the file with the package option `config=<name>`, which will load `<name>.cfg` instead.

```
362 \ifx\bbl@opt@config@nnil
363   \ifpackagewith{babel}{noconfigs}{}%
364   {\InputIfFileExists{bblopts.cfg}%
365     {\typeout{*****^^J%
366       * Local config file bblopts.cfg used^^J%
367       *}}%
368     {}}%
369 \else
```

```

370 \InputIfFileExists{\bbl@opt@config.cfg}%
371   {\typeout{*****^J%
372             * Local config file \bbl@opt@config.cfg used^^J%
373             *}}%
374   {\bbl@error{%
375     Local config file `'\bbl@opt@config.cfg' not found}{%
376     Perhaps you misspelled it.}}%
377 \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 name of the option and the file are the same.

```

378 \bbl@for\bbl@tempa\bbl@language@opts{%
379   \bbl@ifunset{ds@\bbl@tempa}%
380   {\edef\bbl@tempb{%
381     \noexpand\DeclareOption
382     {\bbl@tempa}%
383     {\noexpand\bbl@load@language{\bbl@tempa}}}%
384     \bbl@tempb}%
385     \@empty}

```

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.

```

386 \bbl@foreach\@classoptionslist{%
387   \bbl@ifunset{ds@#1}%
388   {\IfFileExists{#1.ldf}%
389     {\DeclareOption{#1}{\bbl@load@language{#1}}}%
390     {}}%
391   {}

```

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

```

392 \ifx\bbl@opt@main\@nnil\else
393   \expandafter
394   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
395   \DeclareOption{\bbl@opt@main}{}
396 \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  $\LaTeX$  processes before):

```

397 \def\AfterBabelLanguage#1{%
398   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
399 \DeclareOption*{}
400 \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`.

```

401 \ifx\bbl@opt@main\@nnil
402   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
403   \let\bbl@tempc\@empty
404   \bbl@for\bbl@tempb\bbl@tempa{%

```

```

405 \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
406 \ifin@{\edef\bbl@tempc{\bbl@tempb}\fi}
407 \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
408 \expandafter\bbl@tempa\bbl@loaded,\@nnil
409 \ifx\bbl@tempb\bbl@tempc\else
410 \bbl@warning{%
411 Last declared language option is ``\bbl@tempc',\%
412 but the last processed one was ``\bbl@tempb'.\%
413 The main language cannot be set as both a global\%
414 and a package option. Use `main=\bbl@tempc' as\%
415 option. Reported}%
416 \fi
417 \else
418 \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
419 \ExecuteOptions{\bbl@opt@main}
420 \DeclareOption*{}
421 \ProcessOptions*
422 \fi
423 \def\AfterBabelLanguage{%
424 \bbl@error
425 {Too late for \string\AfterBabelLanguage}%
426 {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 `\bbl@main@language`, has become defined. If not, no language has been loaded and an error message is displayed.

```

427 \ifx\bbl@main@language\@undefined
428 \bbl@error{%
429 You haven't specified a language option}{%
430 You need to specify a language, either as a global option\%
431 or as an optional argument to the \string\usepackage\space
432 command;\%
433 You shouldn't try to proceed from here, type x to quit.}
434 \fi
435 \end{package}

```

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

The kernel of the babel system is stored in either `hyphen.cfg` or `switch.def` and `babel.def`. The file `babel.def` contains most of the code, while `switch.def` defines the language switching commands; both can be read at run time. 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 (by default, it also inputs `switch.def`, for “historical reasons”, but it is not necessary). When `babel.def` is loaded it checks if the current version of `switch.def` is in the format; if not it is loaded. A further file, `babel.sty`, contains  $\LaTeX$ -specific stuff. Because plain  $\TeX$  users might want to use some of the features of the babel system too, care has to be taken that plain  $\TeX$  can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain  $\TeX$  and  $\LaTeX$ , some of it is for the  $\LaTeX$  case only. Plain formats based on `etex` (`etex`, `xetex`, `luatex`) don’t load `hyphen.cfg` but `etex.src`, which follows a different naming convention, so we need to define the babel names. It presumes `language.def` exists and it is the same file used when formats were created.

### 8.1 Tools

```

436 \*core)

```

```

437 \ifx\ldf@quit\@undefined
438 \else
439 \expandafter\endinput
440 \fi
441 <<Make sure ProvidesFile is defined>>
442 \ProvidesFile{babel.def}[\<<date>>] [\<<version>>] Babel common definitions]
443 <<Load macros for plain if not LaTeX>>
444 \ifx\bbl@ifshorthand\@undefined
445 \def\bbl@ifshorthand#1#2#3{#2}%
446 \def\bbl@opt@safe{BR}
447 \def\AfterBabelLanguage#1#2{}
448 \let\bbl@afterlang\relax
449 \let\bbl@language@opts\@empty
450 \fi
451 \input switch.def\relax
452 \ifx\bbl@languages\@undefined
453 \ifx\directlua\@undefined
454 \openin1 = language.def
455 \ifeof1
456 \closein1
457 \message{I couldn't find the file language.def}
458 \else
459 \closein1
460 \begingroup
461 \def\addlanguage#1#2#3#4#5{%
462 \expandafter\ifx\csname lang@#1\endcsname\relax\else
463 \global\expandafter\let\csname l@#1\endcsname
464 \csname lang@#1\endcsname
465 \fi}%
466 \def\uselanguage#1{}%
467 \input language.def
468 \endgroup
469 \fi
470 \fi
471 \chardef\l@english\z@
472 \fi
473 <<Load patterns in luatex>>
474 <<Basic macros>>

```

\addto For each language four control sequences have to be defined that control the language-specific definitions. To be able to add something to these macro once they have been defined the macro \addto is introduced. It takes two arguments, a *<control sequence>* and T<sub>E</sub>X-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. Otherwise the replacement text for the *<control sequence>* is expanded and stored in a token register, together with the T<sub>E</sub>X-code to be added. Finally the *<control sequence>* is redefined, using the contents of the token register.

```

475 \def\addto#1#2{%
476 \ifx#1\@undefined
477 \def#1{#2}%
478 \else
479 \ifx#1\relax
480 \def#1{#2}%
481 \else
482 {\toks@\expandafter{#1#2}}%
483 \xdef#1{\the\toks@}%
484 \fi

```

```
485 \fi}
```

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

```
486 \def\bbl@withactive#1#2{%
487 \begingroup
488 \lccode`~=#2\relax
489 \lowercase{\endgroup#1~}}
```

`\bbl@redefine` To redefine a command, we save the old meaning of the macro. Then we redefine it to call the original macro with the ‘sanitized’ argument. The reason why we do it this way is that we don’t want to redefine the  $\LaTeX$  macros completely in case their definitions change (they have changed in the past).

Because we need to redefine a number of commands we define the command `\bbl@redefine` which takes care of this. It creates a new control sequence, `\org@...`

```
490 \def\bbl@redefine#1{%
491 \edef\bbl@tempa{\bbl@stripslash#1}%
492 \expandafter\let\csname org@\bbl@tempa\endcsname#1%
493 \expandafter\def\csname\bbl@tempa\endcsname}
```

This command should only be used in the preamble of the document.

```
494 \@onlypreamble\bbl@redefine
```

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

```
495 \def\bbl@redefine@long#1{%
496 \edef\bbl@tempa{\bbl@stripslash#1}%
497 \expandafter\let\csname org@\bbl@tempa\endcsname#1%
498 \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
499 \@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_`.

```
500 \def\bbl@redefineroobust#1{%
501 \edef\bbl@tempa{\bbl@stripslash#1}%
502 \bbl@ifunset{\bbl@tempa\space}%
503 {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
504 \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
505 {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}%
506 \@namedef{\bbl@tempa\space}}
```

This command should only be used in the preamble of the document.

```
507 \@onlypreamble\bbl@redefineroobust
```

## 8.2 Hooks

Note they are loaded in `babel.def`. `switch.def` only provides a “hook” for hooks (with a default value which is a no-op, below). Admittedly, the current implementation is a somewhat simplistic and does very little to catch errors, but it is intended for developers, after all. `\bbl@usehooks` is the commands used by `babel` to execute hooks defined for an event.

```
508 \def\AddBabelHook#1#2{%
509 \bbl@ifunset{bbl@hk@#1}{\EnableBabelHook{#1}}}%
510 \def\bbl@tempa##1,##2=##2,##3@empty{\def\bbl@tempb{##2}}%
```

```

511 \expandafter\bb1@tempa\bb1@evargs,#2=,\@empty
512 \bb1@ifunset{bb1@ev@#1@#2}%
513 {\bb1@csarg\bb1@add{ev@#2}{\bb1@elt{#1}}}%
514 \bb1@csarg\newcommand}%
515 {\bb1@csarg\let{ev@#1@#2}\relax
516 \bb1@csarg\newcommand}%
517 {ev@#1@#2}[\bb1@tempb]}
518 \def\EnableBabelHook#1{\bb1@csarg\let{hk@#1}\@firstofone}
519 \def\DisableBabelHook#1{\bb1@csarg\let{hk@#1}\@gobble}
520 \def\bb1@usehooks#1#2{%
521 \def\bb1@elt##1{%
522 \@nameuse{bb1@hk@##1}{\@nameuse{bb1@ev@##1@#1}#2}}%
523 \@nameuse{bb1@ev@#1}}

```

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).

```

524 \def\bb1@evargs{,% don't delete the comma
525 everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
526 adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
527 beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
528 hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0}

```

`\babelensure` The user command just parses the optional argument and creates a new macro named `\bb1@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 `\bb1@e@<language>` contains `\bb1@ensure{<include>}{<exclude>}{<fontenc>}`, which in turn loops over the macros names in `\bb1@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.

```

529 \newcommand\babelensure[2][]{% TODO - revise test files
530 \AddBabelHook{babel-ensure}{afterextras}{%
531 \ifcase\bb1@select@type
532 \@nameuse{bb1@e@\languagename}%
533 \fi}%
534 \begingroup
535 \let\bb1@ens@include\@empty
536 \let\bb1@ens@exclude\@empty
537 \def\bb1@ens@fontenc{\relax}%
538 \def\bb1@tempb##1{%
539 \ifx\@empty##1\else\noexpand##1\expandafter\bb1@tempb\fi}%
540 \edef\bb1@tempa{\bb1@tempb#1\@empty}%
541 \def\bb1@tempb##1=##2\@{\@namedef{bb1@ens@##1}{##2}}%
542 \bb1@foreach\bb1@tempa{\bb1@tempb##1\@}%
543 \def\bb1@tempc{\bb1@ensure}%
544 \expandafter\bb1@add\expandafter\bb1@tempc\expandafter{%
545 \expandafter{\bb1@ens@include}}%
546 \expandafter\bb1@add\expandafter\bb1@tempc\expandafter{%
547 \expandafter{\bb1@ens@exclude}}%
548 \toks@\expandafter{\bb1@tempc}%
549 \bb1@exp{%
550 \endgroup
551 \def<bb1@e@#2>{\the\toks@{\bb1@ens@fontenc}}}}
552 \def\bb1@ensure#1#2#3{% 1: include 2: exclude 3: fontenc

```

```

553 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
554   \ifx##1\@empty\else
555     \in@{##1}{#2}%
556     \ifin@ \else
557       \bbl@ifunset{\bbl@ensure@\language}%
558         {\bbl@exp{%
559           \\DeclareRobustCommand\<bbl@ensure@\language>[1]{%
560             \\foreignlanguage{\language}%
561             {\ifx\relax#3\else
562               \\fontencoding{#3}\\selectfont
563               \fi
564               #####1}}}%
565         }%
566       \toks@\expandafter{##1}%
567       \edef##1{%
568         \bbl@csarg\noexpand{ensure@\language}%
569         {\the\toks@}}%
570       \fi
571       \expandafter\bbl@tempb
572     \fi}%
573 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
574 \def\bbl@tempa##1{% elt for include list
575   \ifx##1\@empty\else
576     \bbl@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
577     \ifin@ \else
578       \bbl@tempb##1\@empty
579       \fi
580     \expandafter\bbl@tempa
581     \fi}%
582 \bbl@tempa#1\@empty}
583 \def\bbl@captionslist{%
584 \prefacename\refname\abstractname\bibname\chaptername\appendixname
585 \contentsname\listfigurename\listtablename\indexname\figurename
586 \tablename\partname\enclname\ccname\headtoname\pagename\seename
587 \alsiname\proofname\glossaryname}

```

### 8.3 Setting up language files

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

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

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

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

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

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

Finally we check `\originalTeX`.

```
588 \def\bbl@ldfinit{%
589   \let\bbl@screset@empty
590   \let\BabelStrings\bbl@opt@string
591   \let\BabelOptions@empty
592   \let\BabelLanguages\relax
593   \ifx\originalTeX@undefined
594     \let\originalTeX@empty
595   \else
596     \originalTeX
597   \fi}
598 \def\LdfInit#1#2{%
599   \chardef\atcatcode=\catcode`\  
600   \catcode`\  
601   \chardef\eqcatcode=\catcode`\  
602   \catcode`\  
603   \expandafter\if\expandafter\  
604     \expandafter\  
605     \ifx#2@undefined\else
606       \ldf@quit{#1}%
607     \fi
608   \else
609     \expandafter\ifx\csname#2\endcsname\relax\else
610       \ldf@quit{#1}%
611     \fi
612   \fi
613   \bbl@ldfinit}
```

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

```
614 \def\ldf@quit#1{%
615   \expandafter\main@language\expandafter{#1}%
616   \catcode`\  
617   \catcode`\  
618   \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.

```
619 \def\bbl@afterldf#1{%
620   \bbl@afterlang
621   \let\bbl@afterlang\relax
622   \let\BabelModifiers\relax
623   \let\bbl@screset\relax}%
624 \def\ldf@finish#1{%
625   \loadlocalcfg{#1}%
626   \bbl@afterldf{#1}%
627   \expandafter\main@language\expandafter{#1}%
628   \catcode`\  
629   \catcode`\  
}
```

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  $\LaTeX$ .

```
630 \@onlypreamble\LdfInit
631 \@onlypreamble\ldf@quit
632 \@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.

```
633 \def\main@language#1{%
634   \def\bbl@main@language{#1}%
635   \let\languagename\bbl@main@language
636   \bbl@patterns{\languagename}}
```

We also have to make sure that some code gets executed at the beginning of the document. Languages does not set `\pagedir`, so we set here for the whole document to the main `\bodydir`.

```
637 \AtBeginDocument{%
638   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
639   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place
```

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

```
640 \def\select@language@x#1{%
641   \ifcase\bbl@select@type
642     \bbl@ifsamestring\languagename{#1}{\select@language{#1}}%
643   \else
644     \select@language{#1}%
645   \fi}
```

## 8.4 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.

```
646 \def\bbl@add@special#1% 1:a macro like "\, \?, etc.
647   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
648   \bbl@ifunset{@sanitize}{\bbl@add@sanitize{\@makeother#1}}%
649   \ifx\nfss@catcodes\@undefined\else % TODO - same for above
650     \begingroup
651       \catcode`#1\active
652       \nfss@catcodes
653       \ifnum\catcode`#1=\active
654         \endgroup
655         \bbl@add\nfss@catcodes{\@makeother#1}%
656       \else
657         \endgroup
658       \fi
659   \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.

```
660 \def\bbl@remove@special#1{%
661   \begingroup
662     \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
663       \else\noexpand##1\noexpand##2\fi}%
664     \def\do{\x\do}%
665     \def\@makeother{\x\@makeother}%
666     \edef\x{\endgroup
```

```

667 \def\noexpand\dospecials{\dospecials}%
668 \expandafter\ifx\csname @sanitize\endcsname\relax\else
669 \def\noexpand@sanitize{@sanitize}%
670 \fi}%
671 \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).

```

672 \def\bbl@active@def#1#2#3#4{%
673 \@namedef{#3#1}{%
674 \expandafter\ifx\csname#2@sh@#1\endcsname\relax
675 \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
676 \else
677 \bbl@afterfi\csname#2@sh@#1\endcsname
678 \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.

```

679 \long\@namedef{#3@arg#1}##1{%
680 \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
681 \bbl@afterelse\csname#4#1\endcsname##1%
682 \else
683 \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
684 \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.

```

685 \def\initiate@active@char#1{%
686 \bbl@ifunset{active@char\string#1}%
687 {\bbl@withactive
688 {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
689 {}}

```

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`).

```

690 \def\@initiate@active@char#1#2#3{%

```

```

691 \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
692 \ifx#1\@undefined
693   \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
694 \else
695   \bbl@csarg\let{oridef@#2}#1%
696   \bbl@csarg\edef{oridef@#2}{%
697     \let\noexpand#1%
698     \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
699 \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*).

```

700 \ifx#1#3\relax
701   \expandafter\let\csname normal@char#2\endcsname#3%
702 \else
703   \bbl@info{Making #2 an active character}%
704   \ifnum\mathcode`#2="8000
705     \@namedef{normal@char#2}{%
706       \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
707   \else
708     \@namedef{normal@char#2}{#3}%
709   \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).

```

710   \bbl@restoreactive{#2}%
711   \AtBeginDocument{%
712     \catcode`#2\active
713     \if@filesw
714       \immediate\write\@mainaux{\catcode`\string#2\active}%
715     \fi}%
716   \expandafter\bbl@add@special\csname#2\endcsname
717   \catcode`#2\active
718 \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⟩`).

```

719 \let\bbl@tempa\@firstoftwo
720 \if\string^#2%
721   \def\bbl@tempa{\noexpand\textormath}%
722 \else
723   \ifx\bbl@mathnormal\@undefined\else
724     \let\bbl@tempa\bbl@mathnormal
725   \fi
726 \fi
727 \expandafter\edef\csname active@char#2\endcsname{%

```

```

728 \bbl@tempa
729   {\noexpand\if@safe@actives
730     \noexpand\expandafter
731     \expandafter\noexpand\csname normal@char#2\endcsname
732     \noexpand\else
733     \noexpand\expandafter
734     \expandafter\noexpand\csname bbl@doactive#2\endcsname
735     \noexpand\fi}%
736   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
737 \bbl@csarg\edef{doactive#2}{%
738   \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

$$\backslash\text{active@prefix } \langle char \rangle \backslash\text{normal@char } \langle char \rangle$$

(where  $\backslash\text{active@char } \langle char \rangle$  is *one* control sequence!).

```

739 \bbl@csarg\edef{active#2}{%
740   \noexpand\active@prefix\noexpand#1%
741   \expandafter\noexpand\csname active@char#2\endcsname}%
742 \bbl@csarg\edef{normal#2}{%
743   \noexpand\active@prefix\noexpand#1%
744   \expandafter\noexpand\csname normal@char#2\endcsname}%
745 \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.

```

746 \bbl@active@def#2\user@group{user@active}{language@active}%
747 \bbl@active@def#2\language@group{language@active}{system@active}%
748 \bbl@active@def#2\system@group{system@active}{normal@char}%

```

In order to do the right thing when a shorthand with an argument is used by itself at the end of the line we provide a definition for the case of an empty argument. For that case we let the shorthand character expand to its non-active self. Also, When a shorthand combination such as ' ' ends up in a heading  $\text{T}_\text{E}\text{X}$  would see  $\backslash\text{protect}'\backslash\text{protect}'$ . To prevent this from happening a couple of shorthand needs to be defined at user level.

```

749 \expandafter\edef\csname\user@group @sh#2@@\endcsname
750   {\expandafter\noexpand\csname normal@char#2\endcsname}%
751 \expandafter\edef\csname\user@group @sh#2@\string\protect\endcsname
752   {\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  $\backslash\text{pr@m@s}$  as well. Also, make sure that a single ' in math mode 'does the right thing'. (2) If we are using the caret (^) as a shorthand character special care should be taken to make sure math still works. Therefore an extra level of expansion is introduced with a check for math mode on the upper level.

```

753 \if\string'#2%
754   \let\prim@s\bbl@prim@s
755   \let\active@math@prime#1%
756 \fi
757 \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}

```

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

```

758 <<(*More package options)>> ≡
759 \DeclareOption{math=active}{}
760 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
761 <</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* and the end of the ldf.

```
762 \@ifpackagewith{babel}{KeepShorthandsActive}%
763   {\let\bbl@restoreactive@gobble}%
764   {\def\bbl@restoreactive#1{%
765     \bbl@exp{%
766       \\AfterBabelLanguage\\CurrentOption
767       {\catcode`#1=\the\catcode`#1\relax}%
768       \\AtEndOfPackage
769       {\catcode`#1=\the\catcode`#1\relax}}}%
770   \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.

```
771 \def\bbl@sh@select#1#2{%
772   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
773   \bbl@afterelse\bbl@scndcs
774   \else
775     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
776   \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`.

```
777 \def\active@prefix#1{%
778   \ifx\protect\@typeset@protect
779   \else
```

When `\protect` is set to `\@unexpandable@protect` we make sure that the active character is also *not* expanded by inserting `\noexpand` in front of it. The `\@gobble` is needed to remove a token such as `\activechar:` (when the double colon was the active character to be dealt with).

```
780   \ifx\protect\@unexpandable@protect
781     \noexpand#1%
782   \else
783     \protect#1%
784   \fi
785   \expandafter\@gobble
786   \fi}
```

`\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*).

```
787 \newif\if@safe@actives
788 \@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.

```
789 \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`.

```
790 \def\bbl@activate#1{%
791   \bbl@withactive{\expandafter\let\expandafter}#1%
792   \csname bbl@active@\string#1\endcsname}
793 \def\bbl@deactivate#1{%
794   \bbl@withactive{\expandafter\let\expandafter}#1%
795   \csname bbl@normal@\string#1\endcsname}
```

`\bbl@firstcs` These macros have two arguments. They use one of their arguments to build a control sequence from.

```
796 \def\bbl@firstcs#1#2{\csname#1\endcsname}
797 \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.

```
798 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
799 \def\@decl@short#1#2#3\@nil#4{%
800   \def\bbl@tempa{#3}%
801   \ifx\bbl@tempa\@empty
802     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
803     \bbl@ifunset{#1@sh@\string#2@}{}%
804     {\def\bbl@tempa{#4}%
805      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
806      \else
807        \bbl@info
808        {Redefining #1 shorthand \string#2\%
809         in language \CurrentOption}%
810       \fi}%
811     \@namedef{#1@sh@\string#2@}{#4}%
812   \else
813     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
814     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
815     {\def\bbl@tempa{#4}%
816      \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
817      \else
818        \bbl@info
819        {Redefining #1 shorthand \string#2\string#3\%
820         in language \CurrentOption}%
821       \fi}%
822     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
823   \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.

```
824 \def\textormath{%
825   \ifmmode
826     \expandafter\@secondoftwo
827   \else
828     \expandafter\@firstoftwo
829   \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’.

```

830 \def\user@group{user}
831 \def\language@group{english}
832 \def\system@group{system}

```

`\usesshorthands` This is the user level command to tell  $\LaTeX$  that user level shorthands will be used in the document. It takes one argument, the character that starts a shorthand. First note that this is user level, and then initialize and activate 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.

```

833 \def\usesshorthands{%
834   \ifstar\bb1@usessh{s{\bb1@usessh{x}}}}
835 \def\bb1@usessh@s#1{%
836   \bb1@usessh{x
837     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb1@activate{#1}}}%
838     {#1}}
839 \def\bb1@usessh@x#1#2{%
840   \bb1@ifshorthand{#2}%
841     {\def\user@group{user}%
842       \initiate@active@char{#2}%
843       #1%
844       \bb1@activate{#2}}%
845     {\bb1@error
846       {Cannot declare a shorthand turned off (\string#2)}
847       {Sorry, but you cannot use shorthands which have been\\
848         turned off in the package options}}}

```

`\defineshorthand` Currently we only support two groups of user level shorthands, named internally `user` and `user@<lang>` (language-dependent user shorthands). By default, only the first one is taken into account, but if the former is also used (in the optional argument of `\defineshorthand`) a new level is inserted for it (`user@generic`, done by `\bb1@set@user@generic`); we make also sure `{}` and `\protect` are taken into account in this new top level.

```

849 \def\user@language@group{user@\language@group}
850 \def\bb1@set@user@generic#1#2{%
851   \bb1@ifunset{user@generic@active#1}%
852     {\bb1@active@def#1\user@language@group{user@active}{user@generic@active}%
853       \bb1@active@def#1\user@group{user@generic@active}{language@active}%
854       \expandafter\edef\csname#2@sh@#1@@\endcsname{%
855         \expandafter\noexpand\csname normal@char#1\endcsname}%
856       \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%
857         \expandafter\noexpand\csname user@active#1\endcsname}}%
858   \@empty}
859 \newcommand\defineshorthand[3][user]{%
860   \edef\bb1@tempa{\zap@space#1 \@empty}%
861   \bb1@for\bb1@tempb\bb1@tempa{%
862     \if*\expandafter\@car\bb1@tempb\@nil
863       \edef\bb1@tempb{user@\expandafter\@gobble\bb1@tempb}%
864       \@expandtwoargs
865       \bb1@set@user@generic{\expandafter\string\@car#2\@nil}\bb1@tempb
866     \fi
867     \declare@shorthand{\bb1@tempb}{#2}{#3}}}

```

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

```
868 \def\languageshortands#1{\def\language@group{#1}}
```

`\aliasshorthand` First the new shorthand needs to be initialized,

```
869 \def\aliasshorthand#1#2{%
870   \bbl@ifshorthand{#2}%
871   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
872     \ifx\document\@notprerr
873       \@notshorthand{#2}%
874     \else
875       \initiate@active@char{#2}%
```

Then, we define the new shorthand in terms of the original one, but note with `\aliasshortands{"}{/}` is `\active@prefix /\active@char/`, so we still need to let the latest to `\active@char`.

```
876     \expandafter\let\csname active@char\string#2\endcsname
877     \csname active@char\string#1\endcsname
878     \expandafter\let\csname normal@char\string#2\endcsname
879     \csname normal@char\string#1\endcsname
880     \bbl@activate{#2}%
881   \fi
882 \fi}%
883 {\bbl@error
884   {Cannot declare a shorthand turned off (\string#2)}
885   {Sorry, but you cannot use shortands which have been\\%
886     turned off in the package options}}
```

`\@notshorthand`

```
887 \def\@notshorthand#1{%
888   \bbl@error{%
889     The character '\string #1' should be made a shorthand character;\\%
890     add the command \string\usesshortands\string{#1\string} to
891     the preamble.\\%
892     I will ignore your instruction}%
893   {You may proceed, but expect unexpected results}}
```

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

```
894 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
895 \DeclareRobustCommand*\shorthandoff{%
896   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
897 \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.

```
898 \def\bbl@switch@sh#1#2{%
899   \ifx#2\@nnil\else
900     \bbl@ifunset{bbl@active@\string#2}%
901     {\bbl@error
902       {I cannot switch '\string#2' on or off--not a shorthand}%
903       {This character is not a shorthand. Maybe you made\\%
```

```

904     a typing mistake? I will ignore your instruction}}%
905     {\ifcase#1%
906     \catcode`#212\relax
907     \or
908     \catcode`#2\active
909     \or
910     \csname bbl@oricat@\string#2\endcsname
911     \csname bbl@oridef@\string#2\endcsname
912     \fi}%
913     \bbl@afterfi\bbl@switch@sh#1%
914     \fi}

```

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

```

915 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
916 \def\bbl@putsh#1{%
917     \bbl@ifunset{\bbl@active@\string#1}%
918     {\bbl@putsh@i#1\@empty\@nnil}%
919     {\csname bbl@active@\string#1\endcsname}}
920 \def\bbl@putsh@i#1#2\@nnil{%
921     \csname\languagename @sh@\string#1@%
922     \ifx\@empty#2\else\string#2\fi\endcsname}
923 \ifx\bbl@opt@shorthands\@nnil\else
924     \let\bbl@s@initiate@active@char\initiate@active@char
925     \def\initiate@active@char#1{%
926         \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
927     \let\bbl@s@switch@sh\bbl@switch@sh
928     \def\bbl@switch@sh#1#2{%
929         \ifx#2\@nnil\else
930             \bbl@afterfi
931             \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
932             \fi}
933     \let\bbl@s@activate\bbl@activate
934     \def\bbl@activate#1{%
935         \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
936     \let\bbl@s@deactivate\bbl@deactivate
937     \def\bbl@deactivate#1{%
938         \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
939     \fi

```

`\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.

```

940 \def\bbl@prim@s{%
941     \prime\futurelet\@let@token\bbl@pr@m@s}
942 \def\bbl@if@primes#1#2{%
943     \ifx#1\@let@token
944         \expandafter\@firstoftwo
945     \else\ifx#2\@let@token
946         \bbl@afterelse\expandafter\@firstoftwo
947     \else
948         \bbl@afterfi\expandafter\@secondoftwo
949     \fi\fi}
950 \begingroup
951     \catcode`\^=7 \catcode`\*= \active \lccode`\*= `^
952     \catcode`\'=12 \catcode`\="= \active \lccode`\"= `'
953     \lowercase{%

```

```

954 \gdef\bb1@pr@m@s{%
955 \bb1@if@primes" "%
956 \pr@@@s
957 {\bb1@if@primes*^\pr@@@t\egroup}}
958 \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).

```

959 \initiate@active@char{~}
960 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
961 \bb1@activate{~}

```

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

```

962 \expandafter\def\csname OT1dqpos\endcsname{127}
963 \expandafter\def\csname T1dqpos\endcsname{4}

```

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

```

964 \ifx\f@encoding\@undefined
965 \def\f@encoding{OT1}
966 \fi

```

## 8.5 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.

```

967 \newcommand\languageattribute[2]{%
968 \def\bb1@tempc{#1}%
969 \bb1@fixname\bb1@tempc
970 \bb1@iflanguage\bb1@tempc{%
971 \bb1@vforeach{#2}{%

```

We want to make sure that each attribute is selected only once; therefore we store the already selected attributes in `\bb1@known@attribs`. When that control sequence is not yet defined this attribute is certainly not selected before.

```

972 \ifx\bb1@known@attribs\@undefined
973 \in@false
974 \else

```

Now we need to see if the attribute occurs in the list of already selected attributes.

```

975 \bb1@xin@{\bb1@tempc-##1,}{\bb1@known@attribs,}%
976 \fi

```

When the attribute was in the list we issue a warning; this might not be the users intention.

```

977 \ifin@
978 \bb1@warning{%
979 You have more than once selected the attribute '##1'\%

```

```

980         for language #1}%
981     \else

```

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

```

982     \bbl@exp{%
983         \\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
984     \edef\bbl@tempa{\bbl@tempc-##1}%
985     \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
986     {\csname\bbl@tempc @attr@##1\endcsname}%
987     {\@attrerr{\bbl@tempc}{##1}}%
988     \fi}}

```

This command should only be used in the preamble of a document.

```

989 \@onlypreamble\languageattribute

```

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

```

990 \newcommand*{\@attrerr}[2]{%
991     \bbl@error
992     {The attribute #2 is unknown for language #1.}%
993     {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}`.

```

994 \def\bbl@declare@ttribute#1#2#3{%
995     \bbl@xin@{,#2,}{,\BabelModifiers,}%
996     \ifin@
997     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
998     \fi
999     \bbl@add@list\bbl@attributes{#1-#2}%
1000     \expandafter\def\csname#1@attr@#2\endcsname{#3}}

```

`\bbl@ifattributeset` This internal macro has 4 arguments. It can be used to interpret T<sub>E</sub>X code based on whether a certain attribute was set. This command should appear inside the argument to `\AtBeginDocument` because the attributes are set in the document preamble, *after* babel is loaded.

The first argument is the language, the second argument the attribute being checked, and the third and fourth arguments are the true and false clauses.

```

1001 \def\bbl@ifattributeset#1#2#3#4{%
    First we need to find out if any attributes were set; if not we're done.
1002     \ifx\bbl@known@attribs\@undefined
1003         \in@false
1004     \else

```

The we need to check the list of known attributes.

```

1005     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1006     \fi

```

When we're this far `\ifin@` has a value indicating if the attribute in question was set or not. Just to be safe the code to be executed is 'thrown over the `\fi`'.

```

1007     \ifin@
1008     \bbl@afterelse#3%

```

```

1009 \else
1010 \bbl@afterfi#4%
1011 \fi
1012 }

```

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

```
1013 \def\bbl@ifknown@ttrib#1#2{%
```

We first assume the attribute is unknown.

```
1014 \let\bbl@tempa\@secondoftwo
```

Then we loop over the list of known attributes, trying to find a match.

```

1015 \bbl@loopx\bbl@tempb{#2}{%
1016 \expandafter\in\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1017 \ifin@

```

When a match is found the definition of `\bbl@tempa` is changed.

```

1018 \let\bbl@tempa\@firstoftwo
1019 \else
1020 \fi}%

```

Finally we execute `\bbl@tempa`.

```

1021 \bbl@tempa
1022 }

```

`\bbl@clear@ttribs` This macro removes all the attribute code from L<sup>A</sup>T<sub>E</sub>X's memory at `\begin{document}` time (if any is present).

```

1023 \def\bbl@clear@ttribs{%
1024 \ifx\bbl@attributes\undefined\else
1025 \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1026 \expandafter\bbl@clear@ttrib\bbl@tempa.
1027 }%
1028 \let\bbl@attributes\undefined
1029 \fi}
1030 \def\bbl@clear@ttrib#1-#2.{%
1031 \expandafter\let\csname#1@attr#2\endcsname\undefined}
1032 \AtBeginDocument{\bbl@clear@ttribs}

```

## 8.6 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`

```
1033 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```

1034 \newcount\babel@savecnt
1035 \babel@beginsave

```

`\babel@save` The macro `\babel@save⟨curname⟩` saves the current meaning of the control sequence `⟨curname⟩` to `\originalTeX`<sup>29</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.

```
1036 \def\babel@save#1{%
1037   \expandafter\let\curname babel@\number\babel@savecnt\endcurname#1\relax
1038   \toks@\expandafter{\originalTeX\let#1=}%
1039   \bbl@exp{%
1040     \def\\originalTeX{\the\toks@<babel@\number\babel@savecnt>\relax}}%
1041   \advance\babel@savecnt@ne}
```

`\babel@savevariable` The macro `\babel@savevariable⟨variable⟩` saves the value of the variable. `⟨variable⟩` can be anything allowed after the `\the` primitive.

```
1042 \def\babel@savevariable#1{%
1043   \toks@\expandafter{\originalTeX #1}%
1044   \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 `\bbl@nonfrenchspacing` command `\bbl@frenchspacing` switches it on when it isn't already in effect and `\bbl@nonfrenchspacing` switches it off if necessary.

```
1045 \def\bbl@frenchspacing{%
1046   \ifnum\the\sfcode`. = @m
1047     \let\bbl@nonfrenchspacing\relax
1048   \else
1049     \frenchspacing
1050     \let\bbl@nonfrenchspacing\nonfrenchspacing
1051   \fi}
1052 \let\bbl@nonfrenchspacing\nonfrenchspacing
```

## 8.7 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 `\curname` but the actual macro.

```
1053 \def\babeltags#1{%
1054   \edef\bbl@tempa{\zap@space#1 \@empty}%
1055   \def\bbl@tempb##1=##2\@@{%
1056     \edef\bbl@tempc{%
1057       \noexpand\newcommand
1058       \expandafter\noexpand\curname ##1\endcurname{%
1059         \noexpand\protect
1060         \expandafter\noexpand\curname otherlanguage*\endcurname{##2}}
1061       \noexpand\newcommand
1062       \expandafter\noexpand\curname text##1\endcurname{%
1063         \noexpand\foreignlanguage{##2}}
1064     \bbl@tempc}%
1065   \bbl@for\bbl@tempa\bbl@tempa{%
1066     \expandafter\bbl@tempb\bbl@tempa\@@}}
```

## 8.8 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.

<sup>29</sup>`\originalTeX` has to be expandable, i. e. you shouldn't let it to `\relax`.

See `\bbl@patterns` above for further details. We make sure there is a space between words when multiple commands are used.

```

1067 \onlypreamble\babelhyphenation
1068 \AtEndOfPackage{%
1069   \newcommand\babelhyphenation[2][\@empty]{%
1070     \ifx\bbl@hyphenation\relax
1071       \let\bbl@hyphenation\@empty
1072     \fi
1073     \ifx\bbl@hyphlist\@empty\else
1074       \bbl@warning{%
1075         You must not intermingle \string\selectlanguage\space and\%
1076         \string\babelhyphenation\space or some exceptions will not\%
1077         be taken into account. Reported}%
1078     \fi
1079     \ifx\@empty#1%
1080       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1081     \else
1082       \bbl@vforeach{#1}{%
1083         \def\bbl@tempa{##1}%
1084         \bbl@fixname\bbl@tempa
1085         \bbl@iflanguage\bbl@tempa{%
1086           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1087             \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1088               \@empty
1089               {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1090             #2}}}%
1091       \fi}}

```

`\bbl@allowhyphens` This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak \hskip 0pt plus 0pt`<sup>30</sup>.

```

1092 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1093 \def\bbl@t@one{T1}
1094 \def\allowhyphens{\ifx\cf@encoding\bbl@t@one\else\bbl@allowhyphens\fi}

```

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

```

1095 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1096 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1097 \def\bbl@hyphen{%
1098   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
1099 \def\bbl@hyphen@i#1#2{%
1100   \bbl@ifunset{bbl@hy@#1#2\@empty}%
1101     {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1102     {\csname bbl@hy@#1#2\@empty\endcsname}}

```

The following two commands are used to wrap the “hyphen” and set the behaviour of the rest of the word – the version with a single `@` is used when further hyphenation is allowed, while that with `@@` if no more hyphen 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.

```

1103 \def\bbl@usehyphen#1{%
1104   \leavevmode

```

<sup>30</sup>TeX begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

```

1105 \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1106 \nobreak\hskip\z@skip}
1107 \def\bbl@usehyphen#1{%
1108 \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1109 \def\bbl@hyphenchar{%
1110 \ifnum\hyphenchar\font=\m@ne
1111 \babe\nullhyphen
1112 \else
1113 \char\hyphenchar\font
1114 \fi}

```

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

```

1115 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1116 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1117 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1118 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1119 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1120 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
1121 \def\bbl@hy@repeat{%
1122 \bbl@usehyphen{%
1123 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1124 \def\bbl@hy@repeat{%
1125 \bbl@usehyphen{%
1126 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1127 \def\bbl@hy@empty{\hskip\z@skip}
1128 \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.

```

1129 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#1}\bbl@allowhyphens}

```

## 8.9 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.

```

1130 \def\bbl@tglobal#1{\global\let#1#1}
1131 \def\bbl@recatcode#1{%
1132 \@tempcnta="7F
1133 \def\bbl@tempa{%
1134 \ifnum\@tempcnta>"FF\else
1135 \catcode\@tempcnta=#1\relax
1136 \advance\@tempcnta@ne
1137 \expandafter\bbl@tempa
1138 \fi}%
1139 \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  $\langle\textit{lang}\rangle$ @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).

```
1140 \@ifpackagewith{babel}{nocase}%
1141   {\let\bbl@patchuclc\relax}%
1142   {\def\bbl@patchuclc{%
1143     \global\let\bbl@patchuclc\relax
1144     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1145     \gdef\bbl@uclc##1{%
1146       \let\bbl@encoded\bbl@encoded@uclc
1147       \bbl@ifunset{\languagename @bbl@uclc}% and resumes it
1148       {##1}%
1149       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1150         \csname\languagename @bbl@uclc\endcsname}%
1151       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
1152     \gdef\bbl@tolower{\csname\languagename @bbl@lc\endcsname}%
1153     \gdef\bbl@toupper{\csname\languagename @bbl@uc\endcsname}}%
1154 \langle\langle *More package options \rangle\rangle ≡
1155 \DeclareOption{nocase}{}
1156 \langle\langle /More package options \rangle\rangle
```

The following package options control the behaviour of \SetString.

```
1157 \langle\langle *More package options \rangle\rangle ≡
1158 \let\bbl@opt@strings\@nnil % accept strings=value
1159 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1160 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1161 \def\BabelStringsDefault{generic}
1162 \langle\langle /More package options \rangle\rangle
```

**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.

```
1163 \@onlypreamble\StartBabelCommands
1164 \def\StartBabelCommands{%
1165   \begingroup
1166   \bbl@recatcode{11}%
1167   \langle\langle Macros local to BabelCommands \rangle\rangle
1168   \def\bbl@provstring##1##2{%
1169     \providecommand##1{##2}%
1170     \bbl@tglobal##1}%
1171   \global\let\bbl@scafter\@empty
1172   \let\StartBabelCommands\bbl@startcmds
1173   \ifx\BabelLanguages\relax
1174     \let\BabelLanguages\CurrentOption
1175   \fi
1176   \begingroup
1177   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1178   \StartBabelCommands}
1179 \def\bbl@startcmds{%
1180   \ifx\bbl@screset\@nnil\else
1181     \bbl@usehooks{stopcommands}{}%
1182   \fi
```

```

1183 \endgroup
1184 \begingroup
1185 \@ifstar
1186   {\ifx\bbbl@opt@strings\@nnil
1187     \let\bbbl@opt@strings\BabelStringsDefault
1188     \fi
1189     \bbbl@startcmds@i}%
1190   \bbbl@startcmds@i}
1191 \def\bbbl@startcmds@i#1#2{%
1192   \edef\bbbl@L{\zap@space#1 \@empty}%
1193   \edef\bbbl@G{\zap@space#2 \@empty}%
1194   \bbbl@startcmds@ii}

```

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

Select the behaviour 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.

```

1195 \newcommand\bbbl@startcmds@ii[1][\@empty]{%
1196   \let\SetString\@gobbletwo
1197   \let\bbbl@stringdef\@gobbletwo
1198   \let\AfterBabelCommands\@gobble
1199   \ifx\@empty#1%
1200     \def\bbbl@sc@label{generic}%
1201     \def\bbbl@encstring##1##2{%
1202       \ProvideTextCommandDefault##1{##2}%
1203       \bbbl@tglobal##1%
1204       \expandafter\bbbl@tglobal\csname\string?\string##1\endcsname}%
1205     \let\bbbl@sctest\in@true
1206   \else
1207     \let\bbbl@sc@charset\space % <- zapped below
1208     \let\bbbl@sc@fontenc\space % <- " "
1209     \def\bbbl@tempa##1=##2\@nil{%
1210       \bbbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }%
1211       \bbbl@vforeach{label=#1}{\bbbl@tempa##1\@nil}%
1212       \def\bbbl@tempa##1 ##2{% space -> comma
1213         ##1%
1214         \ifx\@empty##2\else\ifx,##1,\else,\fi\bbbl@afterfi\bbbl@tempa##2\fi}%
1215       \edef\bbbl@sc@fontenc{\expandafter\bbbl@tempa\bbbl@sc@fontenc\@empty}%
1216       \edef\bbbl@sc@label{\expandafter\zap@space\bbbl@sc@label\@empty}%
1217       \edef\bbbl@sc@charset{\expandafter\zap@space\bbbl@sc@charset\@empty}%
1218       \def\bbbl@encstring##1##2{%
1219         \bbbl@foreach\bbbl@sc@fontenc{%
1220           \bbbl@ifunset{T#####1}%
1221           }%
1222           {\ProvideTextCommand##1{#####1}{##2}%
1223           \bbbl@tglobal##1%
1224           \expandafter
1225           \bbbl@tglobal\csname#####1\string##1\endcsname}}}%
1226       \def\bbbl@sctest{%
1227         \bbbl@xin@{\bbbl@opt@strings,}{,\bbbl@sc@label,\bbbl@sc@fontenc,}}%
1228     \fi
1229   \ifx\bbbl@opt@strings\@nnil % ie, no strings key -> defaults

```

```

1230 \else\ifx\bbbl@opt@strings\relax % ie, strings=encoded
1231 \let\AfterBabelCommands\bbbl@aftercmds
1232 \let\SetString\bbbl@setstring
1233 \let\bbbl@stringdef\bbbl@encstring
1234 \else % ie, strings=value
1235 \bbbl@sctest
1236 \ifin@
1237 \let\AfterBabelCommands\bbbl@aftercmds
1238 \let\SetString\bbbl@setstring
1239 \let\bbbl@stringdef\bbbl@provstring
1240 \fi\fi\fi
1241 \bbbl@scswitch
1242 \ifx\bbbl@G\@empty
1243 \def\SetString##1##2{%
1244 \bbbl@error{Missing group for string \string##1}%
1245 {You must assign strings to some category, typically\\%
1246 captions or extras, but you set none}}%
1247 \fi
1248 \ifx\@empty#1%
1249 \bbbl@usehooks{defaultcommands}{}%
1250 \else
1251 \@expandtwoargs
1252 \bbbl@usehooks{encodedcommands}{\bbbl@sc@charset}{\bbbl@sc@fontenc}}%
1253 \fi}

```

There are two versions of `\bbbl@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 (`\bbbl@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 `\bbbl@forlang` loops `\bbbl@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 `\bbbl@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).

```

1254 \def\bbbl@forlang#1#2{%
1255 \bbbl@for#1\bbbl@L{%
1256 \bbbl@xin@{,#1,}{,\BabelLanguages,}%
1257 \ifin@#2\relax\fi}}
1258 \def\bbbl@scswitch{%
1259 \bbbl@forlang\bbbl@tempa{%
1260 \ifx\bbbl@G\@empty\else
1261 \ifx\SetString\@gobbletwo\else
1262 \edef\bbbl@GL{\bbbl@G\bbbl@tempa}%
1263 \bbbl@xin@{,\bbbl@GL,}{,\bbbl@screset,}%
1264 \ifin@\else
1265 \global\expandafter\let\csname\bbbl@GL\endcsname\@undefined
1266 \xdef\bbbl@screset{\bbbl@screset,\bbbl@GL}%
1267 \fi
1268 \fi
1269 \fi}}
1270 \AtEndOfPackage{%
1271 \def\bbbl@forlang#1#2{\bbbl@for#1\bbbl@L{\bbbl@ifunset{date#1}{\#2}}}%
1272 \let\bbbl@scswitch\relax}
1273 \@onlypreamble\EndBabelCommands
1274 \def\EndBabelCommands{%
1275 \bbbl@usehooks{stopcommands}{}%
1276 \endgroup
1277 \endgroup
1278 \bbbl@scafter}

```

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.

```

1279 \def\bbl@setstring#1#2{%
1280   \bbl@forlang\bbl@tempa{%
1281     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1282     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1283     {\global\expandafter % TODO - con \bbl@exp ?
1284       \bbl@add\csname\bbl@G\bbl@tempa\expandafter\endcsname\expandafter
1285         {\expandafter\bbl@scset\expandafter#1\csname\bbl@LC\endcsname}}%
1286     {}}%
1287   \def\BabelString{#2}%
1288   \bbl@usehooks{stringprocess}{}%
1289   \expandafter\bbl@stringdef
1290     \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`.

```

1291 \ifx\bbl@opt@strings\relax
1292   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1293   \bbl@patchuclc
1294   \let\bbl@encoded\relax
1295   \def\bbl@encoded@uclc#1{%
1296     \@inmathwarn#1%
1297     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1298       \expandafter\ifx\csname ?\string#1\endcsname\relax
1299         \TextSymbolUnavailable#1%
1300       \else
1301         \csname ?\string#1\endcsname
1302       \fi
1303     \else
1304       \csname\cf@encoding\string#1\endcsname
1305     \fi}
1306 \else
1307   \def\bbl@scset#1#2{\def#1{#2}}
1308 \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.

```

1309 <<{*Macros local to BabelCommands}>> ≡
1310 \def\SetStringLoop###1##2{%
1311   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1312   \count@\z@
1313   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1314     \advance\count@\@ne
1315     \toks@\expandafter{\bbl@tempa}%
1316     \bbl@exp{%
1317       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1318       \count@=\the\count@\relax}}}%
1319 <</Macros local to BabelCommands>>
```

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

```
1320 \def\bbbl@aftercmds#1{%
1321   \toks@\expandafter{\bbbl@scafter#1}%
1322   \xdef\bbbl@scafter{\the\toks@}}
```

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

```
1323 <<{*Macros local to BabelCommands}>> ≡
1324   \newcommand\SetCase[3][]{%
1325     \bbbl@patchuclc
1326     \bbbl@forlang\bbbl@tempa{%
1327       \expandafter\bbbl@encstring
1328       \csname\bbbl@tempa @bbbl@uclc\endcsname{\bbbl@tempa##1}%
1329       \expandafter\bbbl@encstring
1330       \csname\bbbl@tempa @bbbl@uc\endcsname{##2}%
1331       \expandafter\bbbl@encstring
1332       \csname\bbbl@tempa @bbbl@lc\endcsname{##3}}}%
1333 <</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.

```
1334 <<{*Macros local to BabelCommands}>> ≡
1335   \newcommand\SetHyphenMap[1]{%
1336     \bbbl@forlang\bbbl@tempa{%
1337       \expandafter\bbbl@stringdef
1338       \csname\bbbl@tempa @bbbl@hyphenmap\endcsname{##1}}}%
1339 <</Macros local to BabelCommands>>
```

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

```
1340 \newcommand\BabelLower[2]{% one to one.
1341   \ifnum\lccode#1=#2\else
1342     \babel@savevariable{\lccode#1}%
1343     \lccode#1=#2\relax
1344   \fi}
1345 \newcommand\BabelLowerMM[4]{% many-to-many
1346   \@tempcnta=#1\relax
1347   \@tempcntb=#4\relax
1348   \def\bbbl@tempa{%
1349     \ifnum\@tempcnta>#2\else
1350       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1351       \advance\@tempcnta#3\relax
1352       \advance\@tempcntb#3\relax
1353       \expandafter\bbbl@tempa
1354     \fi}%
1355   \bbbl@tempa}
1356 \newcommand\BabelLowerM0[4]{% many-to-one
1357   \@tempcnta=#1\relax
1358   \def\bbbl@tempa{%
1359     \ifnum\@tempcnta>#2\else
1360       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1361       \advance\@tempcnta#3
1362       \expandafter\bbbl@tempa
1363     \fi}%
1364   \bbbl@tempa}
```

The following package options control the behaviour of hyphenation mapping.

```
1365 <<*More package options>> ≡
1366 \DeclareOption{hyphenmap=off}{\chardef\bb1@opt@hyphenmap\z@}
1367 \DeclareOption{hyphenmap=first}{\chardef\bb1@opt@hyphenmap\@ne}
1368 \DeclareOption{hyphenmap=select}{\chardef\bb1@opt@hyphenmap\tw@}
1369 \DeclareOption{hyphenmap=other}{\chardef\bb1@opt@hyphenmap\thr@@}
1370 \DeclareOption{hyphenmap=other*}{\chardef\bb1@opt@hyphenmap4\relax}
1371 <</More package options>>
```

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

```
1372 \AtEndOfPackage{%
1373   \ifx\bb1@opt@hyphenmap\undefined
1374     \bb1@xin@{,}{\bb1@language@opts}%
1375     \chardef\bb1@opt@hyphenmap\ifin@4\else\@ne\fi
1376   \fi}
```

## 8.10 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.

```
1377 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
1378   \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
1379   \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.

```
1380 \def\save@sf@q#1{\leavevmode
1381   \begingroup
1382   \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
1383   \endgroup}
```

## 8.11 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.

### 8.11.1 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.

```
1384 \ProvideTextCommand{\quotedblbase}{OT1}{%
1385   \save@sf@q{\set@low@box{\textquotedblright\}}%
1386   \box\z@\kern-.04em\bb1@allowhyphens}}
```

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

```
1387 \ProvideTextCommandDefault{\quotedblbase}{%
1388   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
1389 \ProvideTextCommand{\quotesinglbase}{OT1}{%
1390   \save@sf@q{\set@low@box{\textquoteright\}}%
1391   \box\z@\kern-.04em\bb1@allowhyphens}}
```

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

```
1392 \ProvideTextCommandDefault{\quotesinglbase}{%
1393 \UseTextSymbol{OT1}{\quotesinglbase}}
```

`\guillemotleft` The guillemet characters are not available in OT1 encoding. They are faked.  
`\guillemotright`

```
1394 \ProvideTextCommand{\guillemotleft}{OT1}{%
1395 \ifmmode
1396 \ll
1397 \else
1398 \save@sf@q{\nobreak
1399 \raise.2ex\hbox{\scriptscriptstyle\ll}\bbl@allowhyphens}%
1400 \fi}
1401 \ProvideTextCommand{\guillemotright}{OT1}{%
1402 \ifmmode
1403 \gg
1404 \else
1405 \save@sf@q{\nobreak
1406 \raise.2ex\hbox{\scriptscriptstyle\gg}\bbl@allowhyphens}%
1407 \fi}
```

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

```
1408 \ProvideTextCommandDefault{\guillemotleft}{%
1409 \UseTextSymbol{OT1}{\guillemotleft}}
1410 \ProvideTextCommandDefault{\guillemotright}{%
1411 \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
1412 \ProvideTextCommand{\guilsinglleft}{OT1}{%
1413 \ifmmode
1414 <%
1415 \else
1416 \save@sf@q{\nobreak
1417 \raise.2ex\hbox{\scriptscriptstyle<}\bbl@allowhyphens}%
1418 \fi}
1419 \ProvideTextCommand{\guilsinglright}{OT1}{%
1420 \ifmmode
1421 >%
1422 \else
1423 \save@sf@q{\nobreak
1424 \raise.2ex\hbox{\scriptscriptstyle>}\bbl@allowhyphens}%
1425 \fi}
```

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

```
1426 \ProvideTextCommandDefault{\guilsinglleft}{%
1427 \UseTextSymbol{OT1}{\guilsinglleft}}
1428 \ProvideTextCommandDefault{\guilsinglright}{%
1429 \UseTextSymbol{OT1}{\guilsinglright}}
```

### 8.11.2 Letters

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

```
1430 \DeclareTextCommand{\ij}{OT1}{%
1431 i\kern-0.02em\bbl@allowhyphens j}
```

```

1432 \DeclareTextCommand{\IJ}{OT1}{%
1433   \kern-0.02em\bbl@allowhyphens J}
1434 \DeclareTextCommand{\ij}{T1}{\char188}
1435 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

1436 \ProvideTextCommandDefault{\ij}{%
1437   \UseTextSymbol{OT1}{\ij}}
1438 \ProvideTextCommandDefault{\IJ}{%
1439   \UseTextSymbol{OT1}{\IJ}}

```

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

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

```

1440 \def\crrtic@{\hrule height0.1ex width0.3em}
1441 \def\crttic@{\hrule height0.1ex width0.33em}
1442 \def\ddj@{%
1443   \setbox0\hbox{d}\dimen@=\ht0
1444   \advance\dimen@1ex
1445   \dimen@.45\dimen@
1446   \dimen@ii\expandafter\rem@pt\the\fontdimen\@e\font\dimen@
1447   \advance\dimen@ii.5ex
1448   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
1449 \def\DDJ@{%
1450   \setbox0\hbox{D}\dimen@=.55\ht0
1451   \dimen@ii\expandafter\rem@pt\the\fontdimen\@e\font\dimen@
1452   \advance\dimen@ii.15ex % correction for the dash position
1453   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
1454   \dimen\thr@\expandafter\rem@pt\the\fontdimen7\font\dimen@
1455   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
1456 %
1457 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
1458 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

1459 \ProvideTextCommandDefault{\dj}{%
1460   \UseTextSymbol{OT1}{\dj}}
1461 \ProvideTextCommandDefault{\DJ}{%
1462   \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.

```

1463 \DeclareTextCommand{\SS}{OT1}{SS}
1464 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}

```

### 8.11.3 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 1465 \ProvideTextCommandDefault{\glq}{%
1466   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

```

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

```

1467 \ProvideTextCommand{\grq}{T1}{%
1468   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
1469 \ProvideTextCommand{\grq}{TU}{%
1470   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
1471 \ProvideTextCommand{\grq}{OT1}{%
1472   \save@sf@q{\kern-.0125em
1473     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
1474     \kern.07em\relax}}
1475 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}

```

`\glqq` The ‘german’ double quotes.

```

\grqq 1476 \ProvideTextCommandDefault{\glqq}{%
1477   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

1478 \ProvideTextCommand{\grqq}{T1}{%
1479   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1480 \ProvideTextCommand{\grqq}{TU}{%
1481   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1482 \ProvideTextCommand{\grqq}{OT1}{%
1483   \save@sf@q{\kern-.07em
1484     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
1485     \kern.07em\relax}}
1486 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}

```

`\flq` The ‘french’ single guillemets.

```

\frq 1487 \ProvideTextCommandDefault{\flq}{%
1488   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
1489 \ProvideTextCommandDefault{\frq}{%
1490   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

`\flqq` The ‘french’ double guillemets.

```

\frqq 1491 \ProvideTextCommandDefault{\flqq}{%
1492   \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
1493 \ProvideTextCommandDefault{\frqq}{%
1494   \textormath{\guillemotright}{\mbox{\guillemotright}}}

```

#### 8.11.4 Umlauts and tremas

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

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

`\umlautlow`

```

1495 \def\umlauthigh{%
1496   \def\bbl@umlauta##1{\leavevmode\bgroup%
1497     \expandafter\accent\csname\fontencoding dqpos\endcsname
1498     ##1\bbl@allowhyphens\egroup}%
1499   \let\bbl@umlaute\bbl@umlauta}
1500 \def\umlautlow{%
1501   \def\bbl@umlauta{\protect\lower@umlaut}}

```

```

1502 \def\umlaute\lower@umlaut#1{%
1503   \def\bbl@umlaute{\protect\lower@umlaut}}
1504 \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.

```

1505 \expandafter\ifx\csname U@D\endcsname\relax
1506   \csname newdimen\endcsname\U@D
1507 \fi

```

The following code fools T<sub>E</sub>X's `make_accent` procedure about the current x-height of the font to force another placement of the umlaut character. First we have to save the current x-height of the font, because we'll change this font dimension and this is always done globally.

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

```

1508 \def\lower@umlaut#1{%
1509   \leavevmode\bgroup
1510     \U@D 1ex%
1511     {\setbox\z@\hbox{%
1512       \expandafter\char\csname\fontencoding dqpos\endcsname}%
1513       \dimen@ -.45ex\advance\dimen@\ht\z@
1514       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
1515     \expandafter\accent\csname\fontencoding dqpos\endcsname
1516     \fontdimen5\font\U@D #1%
1517   \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).

```

1518 \AtBeginDocument{%
1519   \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
1520   \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
1521   \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{i}}%
1522   \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{i}}%
1523   \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
1524   \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
1525   \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
1526   \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
1527   \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
1528   \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
1529   \DeclareTextCompositeCommand{\"}{OT1}{U}{\bbl@umlauta{U}}%
1530 }

```

Finally, the default is to use English as the main language.

```

1531 \ifx\l@english\undefined
1532   \chardef\l@english\z@
1533 \fi
1534 \main@language{english}

```

## 8.12 Layout

### Work in progress.

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

```
1535 \providecommand\IfBabelLayout[3]{#3}%
1536 \newcommand\BabelPatchSection[1]{%
1537   \ifundefined{#1}{}{%
1538     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
1539     \@namedef{#1}{%
1540       \ifstar{\bbl@presec@#1}%
1541       {\@dblarg{\bbl@presec@#1}}}}
1542 \def\bbl@presec@x#1[#2]#3{%
1543   \bbl@exp{%
1544     \select@language@x{\bbl@main@language}%
1545     \@nameuse{bbl@ss@#1}%
1546     [\select@language{\languagename}{\unexpanded{#2}}]%
1547     {\select@language{\languagename}{\unexpanded{#3}}}%
1548     \select@language@x{\languagename}}
1549 \def\bbl@presec@s#1#2{%
1550   \bbl@exp{%
1551     \select@language@x{\bbl@main@language}%
1552     \@nameuse{bbl@ss@#1}*%
1553     {\select@language{\languagename}{\unexpanded{#2}}}%
1554     \select@language@x{\languagename}}
1555 \IfBabelLayout{sectioning}% at begin document ???
1556 {\BabelPatchSection{part}%
1557  \BabelPatchSection{chapter}%
1558  \BabelPatchSection{section}%
1559  \BabelPatchSection{subsection}%
1560  \BabelPatchSection{subsubsection}%
1561  \BabelPatchSection{paragraph}%
1562  \BabelPatchSection{subparagraph}%
1563  \def\babel@toc#1{%
1564    \select@language@x{\bbl@main@language}}}}
```

Now we load definition files for engines.

```
1565 \ifcase\bbl@engine
1566   \input txtbabel.def
1567 \or
1568   \input luababel.def
1569 \or
1570   \input xebabel.def
1571 \fi
```

## 9 The kernel of Babel (babel.def, only $\LaTeX$ )

### 9.1 The redefinition of the style commands

The rest of the code in this file can only be processed by  $\LaTeX$ , so we check the current format. If it is plain  $\TeX$ , processing should stop here. But, because of the need to limit the scope of the definition of `\format`, a macro that is used locally in the following `\if` statement, this comparison is done inside a group. To prevent  $\TeX$  from complaining about an unclosed group, the processing of the command `\endinput` is deferred until after the group is closed. This is accomplished by the command `\aftergroup`.

```
1572 {\def\format{lplain}
1573 \ifx\fmtname\format
```

```

1574 \else
1575 \def\format{LaTeX2e}
1576 \ifx\fmtname\format
1577 \else
1578 \aftergroup\endinput
1579 \fi
1580 \fi}

```

## 9.2 Creating languages

`\babelprovide` is a general purpose tool for creating languages. Currently it just creates the language infrastructure, but in the future it will be able to read data from ini files, as well as to create variants. Unlike the nil pseudo-language, captions are defined, but with a warning to invite the user to provide the real string.

```

1581 \newcommand\babelprovide[2][{}%
1582 \let\bb1@savelangname\languagename
1583 \def\languagename{#2}%
1584 \let\bb1@KVP@captions\@nil
1585 \let\bb1@KVP@import\@nil
1586 \let\bb1@KVP@main\@nil
1587 \let\bb1@KVP@script\@nil
1588 \let\bb1@KVP@language\@nil
1589 \let\bb1@KVP@dir\@nil
1590 \let\bb1@KVP@hyphenrules\@nil
1591 \bb1@forkv{#1}{\bb1@csarg\def{KVP@##1}{##2}}% TODO - error handling
1592 \ifx\bb1@KVP@captions\@nil
1593 \let\bb1@KVP@captions\bb1@KVP@import
1594 \fi
1595 \bb1@ifunset{date#2}%
1596 {\bb1@provide@new{#2}}%
1597 {\bb1@ifblank{#1}%
1598 {\bb1@error
1599 {If you want to modify `#2' you must tell how in\\
1600 the optional argument. Currently there are three\\
1601 options: captions=lang-tag, hyphenrules=lang-list\\
1602 import=lang-tag}%
1603 {Use this macro as documented}}%
1604 {\bb1@provide@renew{#2}}}%
1605 \bb1@exp{\babelensure[exclude=\\today]{#2}}%
1606 \bb1@ifunset{bb1@ensure@\languagename}%
1607 {\bb1@exp{%
1608 \\\DeclareRobustCommand\<bb1@ensure@\languagename>[1]{%
1609 \\\foreignlanguage{\languagename}%
1610 {###1}}}%
1611 }%
1612 \ifx\bb1@KVP@script\@nil\else
1613 \bb1@csarg\edef{sname#2}{\bb1@KVP@script}%
1614 \fi
1615 \ifx\bb1@KVP@language\@nil\else
1616 \bb1@csarg\edef{lname#2}{\bb1@KVP@language}%
1617 \fi
1618 \let\languagename\bb1@savelangname}

```

Depending on whether or not the language exists, we define two macros.

```

1619 \def\bb1@provide@new#1{%
1620 \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
1621 \@namedef{extras#1}{}%
1622 \@namedef{noextras#1}{}%

```

```

1623 \StartBabelCommands*{#1}{captions}%
1624 \ifx\bb1@KVP@captions\@nil % and also if import, implicit
1625 \def\bb1@tempb##1{% elt for \bb1@captionslist
1626 \ifx##1\@empty\else
1627 \bb1@exp{%
1628 \\\SetString\\##1{%
1629 \\\bb1@nocaption{\bb1@stripslash##1}{\<#1\bb1@stripslash##1>}}}%
1630 \expandafter\bb1@tempb
1631 \fi}%
1632 \expandafter\bb1@tempb\bb1@captionslist\@empty
1633 \else
1634 \bb1@read@ini{\bb1@KVP@captions}% Here all letters cat = 11
1635 \bb1@after@ini
1636 \bb1@savestrings
1637 \fi
1638 \StartBabelCommands*{#1}{date}%
1639 \ifx\bb1@KVP@import\@nil
1640 \bb1@exp{%
1641 \\\SetString\\today{\bb1@nocaption{today}{\<#1today>}}}%
1642 \else
1643 \bb1@savetoday
1644 \bb1@savedate
1645 \fi
1646 \EndBabelCommands
1647 \bb1@exp{%
1648 \def\<#1hyphenmins>{%
1649 {\bb1@ifunset{\bb1@lfthm@#1}{2}{\@nameuse{\bb1@lfthm@#1}}}%
1650 {\bb1@ifunset{\bb1@rgthm@#1}{3}{\@nameuse{\bb1@rgthm@#1}}}}}%
1651 \bb1@provide@hyphens{#1}%
1652 \ifx\bb1@KVP@main\@nil\else
1653 \expandafter\main@language\expandafter{#1}%
1654 \fi}
1655 \def\bb1@provide@renew#1{%
1656 \ifx\bb1@KVP@captions\@nil\else
1657 \StartBabelCommands*{#1}{captions}%
1658 \bb1@read@ini{\bb1@KVP@captions}% Here all letters cat = 11
1659 \bb1@after@ini
1660 \bb1@savestrings
1661 \EndBabelCommands
1662 \fi
1663 \ifx\bb1@KVP@import\@nil\else
1664 \StartBabelCommands*{#1}{date}%
1665 \bb1@savetoday
1666 \bb1@savedate
1667 \EndBabelCommands
1668 \fi
1669 \bb1@provide@hyphens{#1}}

```

The hyphenrules option is handled with an auxiliary macro.

```

1670 \def\bb1@provide@hyphens#1{%
1671 \let\bb1@tempa\relax
1672 \ifx\bb1@KVP@hyphenrules\@nil\else
1673 \bb1@replace\bb1@KVP@hyphenrules{ }{,}%
1674 \bb1@foreach\bb1@KVP@hyphenrules{%
1675 \ifx\bb1@tempa\relax % if not yet found
1676 \bb1@ifsamestring{##1}{+}%
1677 {\bb1@exp{\addlanguage\<1@##1>}}}%
1678 {}%
1679 \bb1@ifunset{1@##1}%

```

```

1680         {}%
1681         {\bbl@exp{\let\bbl@tempa\<l@##1>}}%
1682     \fi}%
1683 \fi
1684 \ifx\bbl@tempa\relax %           if no opt or no language in opt found
1685     \ifx\bbl@KVP@import\@nil\else % if importing
1686         \bbl@exp{%               and hyphenrules is not empty
1687             \\bbl@ifblank{\@nameuse{bbl@hyphr@#1}}%
1688             {}%
1689             {\let\\bbl@tempa\<l@\@nameuse{bbl@hyphr@\language}\>}}%
1690     \fi
1691 \fi
1692 \bbl@ifunset{bbl@tempa}%         ie, relax or undefined
1693     {\bbl@ifunset{l@#1}%         no hyphenrules found - fallback
1694         {\bbl@exp{\\adddialect\<l@#1>\language}}%
1695         {}%                     so, l@<lang> is ok - nothing to do
1696     {\bbl@exp{\\adddialect\<l@#1>\bbl@tempa}}% found in opt list or ini

```

The reader of ini files. There are 3 possible cases: a section name (in the form [ . . . ]), a comment (starting with ; ) and a key/value pair. *TODO - Work in progress.*

```

1697 \def\bbl@read@ini#1{%
1698     \openin1=babel-#1.ini
1699     \ifeof1
1700         \bbl@error
1701         {There is no ini file for the requested language\\%
1702             (#1). Perhaps you misspelled it or your installation\\%
1703             is not complete.}%
1704         {Fix the name or reinstall babel.}%
1705     \else
1706         \let\bbl@section\@empty
1707         \let\bbl@savestrings\@empty
1708         \let\bbl@savetoday\@empty
1709         \let\bbl@savestate\@empty
1710         \let\bbl@inireader\bbl@iniskip
1711         \bbl@info{Importing data from babel-#1.ini for \language}%
1712         \loop
1713         \if T\ifeof1F\fi T\relax % Trick, because inside \loop
1714             \endlinechar\m@ne
1715             \read1 to \bbl@line
1716             \endlinechar\^^M
1717             \ifx\bbl@line\@empty\else
1718                 \expandafter\bbl@iniline\bbl@line\bbl@iniline
1719             \fi
1720         \repeat
1721     \fi}
1722 \def\bbl@iniline#1\bbl@iniline{%
1723     \@ifnextchar[\bbl@inisec{\@ifnextchar;\bbl@iniskip\bbl@inireader}#1\@}% ]

```

The special cases for comment lines and sections are handled by the two following commands. In sections, we provide the possibility to take extra actions at the end or at the start (TODO - but note the last section is not ended). By default, key=val pairs are ignored.

```

1724 \def\bbl@iniskip#1\@{%         if starts with ;
1725 \def\bbl@inisec[#1]#2\@{%     if starts with opening bracket
1726     \@nameuse{bbl@secpst@\bbl@section}% ends previous section
1727     \def\bbl@section{#1}%
1728     \@nameuse{bbl@secpre@\bbl@section}% starts current section
1729     \bbl@ifunset{bbl@secline@#1}%
1730     {\let\bbl@inireader\bbl@iniskip}%
1731     {\bbl@exp{\let\\bbl@inireader\<bbl@secline@#1>}}

```

Reads a key=val line and stores the trimmed val in \bbl@kv@<section>.<key>.

```
1732 \def\bbl@inikv#1=#2\@@{%      key=value
1733 \bbl@trim@def\bbl@tempa{#1}%
1734 \bbl@trim\toks@{#2}%
1735 \bbl@csarg\edef{kv@\bbl@section.\bbl@tempa}{\the\toks@}}
```

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

```
1736 \def\bbl@exportkey#1#2#3{%
1737 \bbl@ifunset{bbl@kv@#2}%
1738 { \bbl@csarg\gdef{#1@\languagename}{#3}}%
1739 { \expandafter\ifx\csname bbl@kv@#2\endcsname\empty
1740 \bbl@csarg\gdef{#1@\languagename}{#3}}%
1741 \else
1742 \bbl@exp{\global\let\<bbl@#1@\languagename>\<bbl@kv@#2>}%
1743 \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.

```
1744 \let\bbl@secline@identification\bbl@inikv
1745 \def\bbl@secpost@identification{%
1746 \bbl@exportkey{lname}{identification.name.english}{}%
1747 \bbl@exportkey{lbcpr}{identification.tag.bcp47}{}%
1748 \bbl@exportkey{lotf}{identification.tag.opentype}{dfmt}%
1749 \bbl@exportkey{sname}{identification.script.name}{}%
1750 \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
1751 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}}
1752 \let\bbl@secline@typography\bbl@inikv
1753 \def\bbl@after@ini{%
1754 \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
1755 \bbl@exportkey{rgtm}{typography.righthyphenmin}{3}%
1756 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
1757 \def\bbl@tempa{0.9}%
1758 \bbl@csarg\ifx{kv@identification.version}\bbl@tempa
1759 \bbl@warning{%
1760 The '\languagename' date format may not be suitable\%
1761 for proper typesetting, and therefore it very likely will\%
1762 change in a future release. Reported}%
1763 \fi
1764 \bbl@tglobal\bbl@savetoday
1765 \bbl@tglobal\bbl@savestate}
```

Now captions and captions.licr, depending on the engine. And also for dates. They rely on a few auxiliary macros.

```
1766 \ifcase\bbl@engine
1767 \bbl@csarg\def{secline@captions.licr}#1=#2\@@{%
1768 \bbl@ini@captions@aux{#1}{#2}}
1769 \bbl@csarg\def{secline@date.gregorian}#1=#2\@@{%      for defaults
1770 \bbl@ini@dategreg#1...\relax{#2}}
1771 \bbl@csarg\def{secline@date.gregorian.licr}#1=#2\@@{%  override
1772 \bbl@ini@dategreg#1...\relax{#2}}
1773 \else
1774 \def\bbl@secline@captions#1=#2\@@{%
1775 \bbl@ini@captions@aux{#1}{#2}}
1776 \bbl@csarg\def{secline@date.gregorian}#1=#2\@@{%
1777 \bbl@ini@dategreg#1...\relax{#2}}
1778 \fi
```

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

```

1779 \def\bbl@ini@captions@aux#1#2{%
1780 \bbl@trim@def\bbl@tempa{#1}%
1781 \bbl@ifblank{#2}%
1782 {\bbl@exp{%
1783 \toks@{\bbl@nocaption{\bbl@tempa}\<language name\bbl@tempa name>}}}%
1784 {\bbl@trim\toks@{#2}}%
1785 \bbl@exp{%
1786 \bbl@add\bbl@savestrings{%
1787 \SetString\<bbl@tempa name>{\the\toks@}}}
```

But dates are more complex. The full date format is stores in date.gregorian, so we must read it in non-Unicode engines, too.

```

1788 \bbl@csarg\def{secre@date.gregorian.licr}{%
1789 \ifcase\bbl@engine\let\bbl@savestate@empty\fi}
1790 \def\bbl@ini@dategreg#1.#2.#3.#4\relax#5{% TODO - ignore with 'captions'
1791 \bbl@trim@def\bbl@tempa{#1.#2}%
1792 \bbl@ifsamestring{\bbl@tempa}{months.wide}%
1793 {\bbl@trim@def\bbl@tempa{#3}%
1794 \bbl@trim\toks@{#5}%
1795 \bbl@exp{%
1796 \bbl@add\bbl@savestate{%
1797 \SetString\<month\romannumeral\bbl@tempa name>{\the\toks@}}}%
1798 {\bbl@ifsamestring{\bbl@tempa}{date.long}%
1799 {\bbl@trim@def\bbl@toreplace{#5}%
1800 \bbl@TG@date
1801 \global\bbl@csarg\let{date@\<language name>}\bbl@toreplace
1802 \bbl@exp{%
1803 \gdef\<language name date>{\protect\<language name date >}}%
1804 \gdef\<language name date >###1###2###3{%
1805 \bbl@usedategrouptrue
1806 \<bbl@ensure@\<language name>}{%
1807 \<bbl@date@\<language name>{###1}{###2}{###3}}}%
1808 \bbl@add\bbl@savetoday{%
1809 \SetString\<<today>{%
1810 \<language name date>{\the\year}{\the\month}{\the\day}}}%
1811 {}}
```

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.

```

1812 \newcommand\BabelDateSpace{\nobreakspace}
1813 \newcommand\BabelDateDot{.\@}
1814 \newcommand\BabelDated[1]{\number#1}
1815 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
1816 \newcommand\BabelDateM[1]{\number#1}
1817 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
1818 \newcommand\BabelDateMMMM[1]{%
1819 \csname month\romannumeral#1name\endcsname}%
1820 \newcommand\BabelDatey[1]{\number#1}%
1821 \newcommand\BabelDateyy[1]{%
1822 \ifnum#1<10 0\number#1 %
1823 \else\ifnum#1<100 \number#1 %
1824 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
1825 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
1826 \else
1827 \bbl@error
1828 {Currently two-digit years are restricted to the\}}
```

```

1829     range 0-9999.}%
1830     {There is little you can do. Sorry.}%
1831     \fi\fi\fi\fi}}
1832 \newcommand\BabelDateyyy[1]{\number#1}}
1833 \def\bbl@replace@finish@iii#1{%
1834   \bbl@exp{\def\#1###1###2###3{\the\toks@}}
1835 \def\bbl@TG@date{%
1836   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
1837   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
1838   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{###3}}%
1839   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{###3}}%
1840   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{###2}}%
1841   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{###2}}%
1842   \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{###2}}%
1843   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{###1}}%
1844   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{###1}}%
1845   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{###1}}%
1846 % Note after \bbl@replace \toks@ contains the resulting string.
1847 % TODO - Using this implicit behavior doesn't seem a good idea.
1848   \bbl@replace@finish@iii\bbl@toreplace}

```

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

```

1849 \def\bbl@provide@lsys#1{%
1850   \bbl@ifunset{bbl@lname@#1}%
1851     {\bbl@ini@ids{#1}}%
1852     {}%
1853   \bbl@csarg\let{lsys@#1}\@empty
1854   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
1855   \bbl@ifunset{bbl@sotf#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
1856   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
1857   \bbl@ifunset{bbl@lname@#1}{%
1858     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
1859   \bbl@csarg\bbl@to@global{lsys@#1}}%
1860 % \bbl@exp{% TODO - should be global
1861 %   \<keys_if_exist:nnF>{fontspec-opentype/Script}{\bbl@cs{sname@#1}}%
1862 %   {\newfontscript{\bbl@cs{sname@#1}}{\bbl@cs{sotf@#1}}}%
1863 %   \<keys_if_exist:nnF>{fontspec-opentype/Language}{\bbl@cs{lname@#1}}%
1864 %   {\newfontlanguage{\bbl@cs{lname@#1}}{\bbl@cs{lotf@#1}}}}

```

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.

```

1865 \def\bbl@ini@ids#1{%
1866   \def\BabelBeforeIni###2{%
1867     \begingroup
1868       \bbl@add\bbl@secpost@identification{\closein1 }%
1869       \catcode`\[=12 \catcode`\]=12 \catcode`\=12
1870       \bbl@read@ini{##1}%
1871     \endgroup}% boxed, to avoid extra spaces:
1872   {\setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}}}}

```

### 9.3 Cross referencing macros

The  $\LaTeX$  book states:

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

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

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

The only way to accomplish this in most cases is to use the trick described in the *T<sub>E</sub>Xbook* [2] (Appendix D, page 382). The primitive `\meaning` applied to a token expands to the current meaning of this token. For example, `\meaning\A` with `\A` defined as `\def\A#1{\B}` expands to the characters ‘macro:#1->\B’ with all category codes set to ‘other’ or ‘space’.

`\newlabel` The macro `\label` writes a line with a `\newlabel` command into the `.aux` file to define labels.

```
1873 %\bbl@redefine\newlabel#1#2{%
1874 % \@safe@activestrue\org@newlabel{#1}{#2}\@safe@activesfalse}
```

`\@newl@bel` We need to change the definition of the *L<sup>A</sup>T<sub>E</sub>X*-internal macro `\@newl@bel`. This is needed because we need to make sure that shorthand characters expand to their non-active version.

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

```
1875 <<{*More package options}>> ≡
1876 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
1877 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
1878 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
1879 <</More package options>>
```

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.

```
1880 \ifx\bbl@opt@safe\@empty\else
1881 \def\@newl@bel#1#2#3{%
1882   {\@safe@activestrue
1883     \bbl@ifunset{#1#2}%
1884     \relax
1885     {\gdef\@multiplelabels%
1886       \@latex@warning@no@line{There were multiply-defined labels}}%
1887     \@latex@warning@no@line{Label `#2' multiply defined}}%
1888   \global\@namedef{#1#2}{#3}}
```

`\@testdef` An internal *L<sup>A</sup>T<sub>E</sub>X* macro used to test if the labels that have been written on the `.aux` file have changed. It is called by the `\enddocument` macro. This macro needs to be completely rewritten, using `\meaning`. The reason for this is that in some cases the expansion of `\#1@#2` contains the same characters as the `#3`; but the character codes differ. Therefore *L<sup>A</sup>T<sub>E</sub>X* keeps reporting that the labels may have changed.

```
1889 \CheckCommand*\@testdef[3]{%
1890   \def\reserved@a{#3}%
1891   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
1892   \else
1893     \@tempwattrue
1894   \fi}
```

Now that we made sure that `\@testdef` still has the same definition we can rewrite it. First we make the shorthands ‘safe’.

```
1895 \def\@testdef#1#2#3{%
1896   \@safe@activestrue
```

Then we use `\bbl@tempa` as an ‘alias’ for the macro that contains the label which is being checked.

```
1897 \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
```

Then we define `\bbl@tempb` just as `\@newlabel` does it.

```
1898 \def\bbl@tempb{#3}%
```

```
1899 \@safe@activesfalse
```

When the label is defined we replace the definition of `\bbl@tempa` by its meaning.

```
1900 \ifx\bbl@tempa\relax
```

```
1901 \else
```

```
1902 \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
```

```
1903 \fi
```

We do the same for `\bbl@tempb`.

```
1904 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
```

If the label didn’t change, `\bbl@tempa` and `\bbl@tempb` should be identical macros.

```
1905 \ifx\bbl@tempa\bbl@tempb
```

```
1906 \else
```

```
1907 \@tempswatrue
```

```
1908 \fi}
```

```
1909 \fi
```

`\ref` `\pageref` The same holds for the macro `\ref` that references a label and `\pageref` to reference a page. So we redefine `\ref` and `\pageref`. While we change these macros, we make them robust as well (if they weren’t already) to prevent problems if they should become expanded at the wrong moment.

```
1910 \bbl@xin@{R}\bbl@opt@safe
```

```
1911 \ifin@
```

```
1912 \bbl@redefineroast\ref#1{%
```

```
1913 \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
```

```
1914 \bbl@redefineroast\pageref#1{%
```

```
1915 \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
```

```
1916 \else
```

```
1917 \let\org@ref\ref
```

```
1918 \let\org@pageref\pageref
```

```
1919 \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.

```
1920 \bbl@xin@{B}\bbl@opt@safe
```

```
1921 \ifin@
```

```
1922 \bbl@redefine\@citex[#1]#2{%
```

```
1923 \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
```

```
1924 \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.

```
1925 \AtBeginDocument{%
```

```
1926 \@ifpackageloaded{natbib}{%
```



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

```
1948 \AtBeginDocument{\bbl@cite@choice}
```

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

```
1949 \bbl@redefine\@bibitem#1{%
1950   \@safe@activestrue\org@bibitem{#1}\@safe@activesfalse}
1951 \else
1952   \let\org@nocite\nocite
1953   \let\org@citex\citex
1954   \let\org@babcite\babcite
1955   \let\org@bibitem\@bibitem
1956 \fi
```

## 9.4 Marks

`\markright` Because the output routine is asynchronous, we must pass the current language attribute to the head lines, together with the text that is put into them. To achieve this we need to adapt the definition of \markright and \markboth somewhat. We check whether the argument is empty; if it is, we just make sure the scratch token register is empty. Next, we store the argument to \markright in the scratch token register. This way these commands will not be expanded later, and we make sure that the text is typeset using the correct language settings. While doing so, we make sure that active characters that may end up in the mark are not disabled by the output routine kicking in while \@safe@activestrue is in effect.

```
1957 \IfBabelLayout{sectioning}
1958   {\ifx\bbl@opt@headfoot\@nnil
1959     \g@addto@macro\@resetactivechars{%
1960       \set@typeset@protect
1961       \expandafter\select@language@x\expandafter{\bbl@main@language}%
1962       \let\protect\noexpand}%
1963   \fi}
1964   {\bbl@redefine\markright#1{%
1965     \bbl@ifblank{#1}%
1966     {\org@markright{}}%
1967     {\toks@{#1}%
1968     \bbl@exp{%
1969       \\org@markright{\\protect\\foreignlanguage{\language}%
1970       {\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.

```
\@mkboth
1971   \ifx\@mkboth\markboth
1972     \def\bbl@tempc{\let\@mkboth\markboth}
1973   \else
1974     \def\bbl@tempc{}
1975   \fi
```

Now we can start the new definition of \markboth

```
1976 \bbl@redefine\markboth#1#2{%
1977   \protected@edef\bbl@tempb##1{%
1978     \protect\foreignlanguage{\language}{\protect\bbl@restore@actives##1}}%
```

```

1979     \bbl@ifblank{#1}%
1980         {\toks@{}}%
1981     {\toks\expandafter{\bbl@tempb{#1}}}%
1982     \bbl@ifblank{#2}%
1983         {\@temptokena{}}%
1984     {\@temptokena\expandafter{\bbl@tempb{#2}}}%
1985     \bbl@exp{\@org@markboth{\the\toks@}{\the\@temptokena}}
    and copy it to \@mkboth if necessary.
1986     \bbl@tempc} % end \IfBabelLayout

```

## 9.5 Preventing clashes with other packages

### 9.5.1 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.

The first thing we need to do is check if the package `ifthen` is loaded. This should be done at `\begin{document}` time.

```

1987 \bbl@xin@{R}\bbl@opt@safe
1988 \ifin@
1989     \AtBeginDocument{%
1990         \@ifpackageloaded{ifthen}{%

```

Then we can redefine `\ifthenelse`:

```

1991     \bbl@redefine@long\ifthenelse#1#2#3{%

```

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.

```

1992         \let\bbl@temp@pref\pageref
1993         \let\pageref\org@pageref
1994         \let\bbl@temp@ref\ref
1995         \let\ref\org@ref

```

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. When the package wasn't loaded we do nothing.

```

1996         \@safe@activestrue
1997         \org@ifthenelse{#1}%
1998             {\let\pageref\bbl@temp@pref
1999             \let\ref\bbl@temp@ref
2000             \@safe@activesfalse
2001             #2}%
2002             {\let\pageref\bbl@temp@pref
2003             \let\ref\bbl@temp@ref
2004             \@safe@activesfalse
2005             #3}%
2006     }%
2007 }{}%
2008 }

```

### 9.5.2 varioref

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

```
\Ref 2009 \AtBeginDocument{%
2010   \@ifpackageloaded{varioref}{%
2011     \bbl@redefine\@vpageref#1[#2]#3{%
2012       \@safe@activestrue
2013       \org@@vpageref{#1}[#2]{#3}%
2014       \@safe@activesfalse}%
```

The same needs to happen for `\vrefpagenum`.

```
2015   \bbl@redefine\vrefpagenum#1#2{%
2016     \@safe@activestrue
2017     \org@vrefpagenum{#1}{#2}%
2018     \@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.

```
2019   \expandafter\def\csname Ref \endcsname#1{%
2020     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
2021   }{}%
2022 }
2023 \fi
```

### 9.5.3 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.

So at `\begin{document}` we check whether `hhline` is loaded.

```
2024 \AtEndOfPackage{%
2025   \AtBeginDocument{%
2026     \@ifpackageloaded{hhline}%
```

Then we check whether the expansion of `\normal@char:` is not equal to `\relax`.

```
2027     {\expandafter\ifx\csname normal@char:string:\endcsname\relax
2028       \else
```

In that case we simply reload the package. 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.

```
2029       \makeatletter
2030       \def\@currname{hhline}\input{hhline.sty}\makeatother
2031       \fi}%
2032     {}}}
```

### 9.5.4 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.

```

2033 \AtBeginDocument{%
2034   \ifx\pdfstringdefDisableCommands\undefined\else
2035     \pdfstringdefDisableCommands{\languageshortands\system}}%
2036   \fi}

```

### 9.5.5 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.

```

2037 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
2038   \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.

```

2039 \def\substitutefontfamily#1#2#3{%
2040   \lowercase{\immediate\openout15=#1#2.fd\relax}%
2041   \immediate\write15{%
2042     \string\ProvidesFile{#1#2.fd}%
2043     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
2044     \space generated font description file]^^J
2045     \string\DeclareFontFamily{#1}{#2}{}^^J
2046     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
2047     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
2048     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
2049     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
2050     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
2051     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
2052     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
2053     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
2054   }%
2055   \closeout15
2056 }

```

This command should only be used in the preamble of a document.

```

2057 \@onlypreamble\substitutefontfamily

```

## 9.6 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. 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`

```

2058 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,}
2059 \let\org@TeX\TeX
2060 \let\org@LaTeX\LaTeX
2061 \let\ensureascii\@firstofone
2062 \AtBeginDocument{%
2063   \in@false
2064   \bbl@foreach\BabelNonASCII{% is there a non-ascii enc?

```

```

2065 \ifin@ \else
2066 \lowercase{\bbl@xin@{,#1enc.def,}{, \@filelist,}}%
2067 \fi}%
2068 \ifin@ % if a non-ascii has been loaded
2069 \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
2070 \DeclareTextCommandDefault{\TeX}{\org@TeX}%
2071 \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
2072 \def\bbl@tempb#1\@@{\uppercase{\bbl@tempc#1}ENC.DEF\@empty\@@}%
2073 \def\bbl@tempc#1ENC.DEF#2\@@{%
2074 \ifx\@empty#2\else
2075 \bbl@ifunset{T#1}%
2076 }%
2077 {\bbl@xin@{,#1,}{, \BabelNonASCII,}}%
2078 \ifin@
2079 \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
2080 \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
2081 \else
2082 \def\ensureascii##1{{\fontencoding{#1}\selectfont##1}}%
2083 \fi}%
2084 \fi}%
2085 \bbl@foreach\@filelist{\bbl@tempb#1\@@}% TODO - \@ de mas??
2086 \bbl@xin@{,\cf@encoding,}{, \BabelNonASCII,}%
2087 \ifin@ \else
2088 \edef\ensureascii#1{%
2089 \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
2090 \fi
2091 \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.

```

2092 \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.

```

2093 \AtBeginDocument{%
2094 \ifpackageloaded{fontspec}%
2095 {\xdef\latinencoding{%
2096 \ifx\UTFencname\undefined
2097 EU\ifcase\bbl@engine\or2\or1\fi
2098 \else
2099 \UTFencname
2100 \fi}}%
2101 {\gdef\latinencoding{OT1}%
2102 \ifx\cf@encoding\bbl@t@one
2103 \xdef\latinencoding{\bbl@t@one}%
2104 \else
2105 \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}}%
2106 \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.

```

2107 \DeclareRobustCommand{\latintext}{%
2108   \fontencoding{\latinencoding}\selectfont
2109   \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.

```

2110 \ifx\@undefined\DeclareTextFontCommand
2111   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
2112 \else
2113   \DeclareTextFontCommand{\textlatin}{\latintext}
2114 \fi

```

## 9.7 Basic bidi support

**Work in progress.** This code is currently placed here for practical reasons.

- 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 T<sub>E</sub>X 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 LuaT<sub>E</sub>X-ja shows, vertical typesetting is possible, too. Its main drawback is font handling is often considered to be less mature than xetex, mainly in Indic scripts (but there are steps to make HarfBuzz, the xetex font engine, available in luatex; see <<https://github.com/tatzetwerk/luatex-harfbuzz>>).

```

2115 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
2116 \def\bbl@rscripts{%
2117   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
2118   Old Hungarian,Old Hungarian,Lybian,Mandaean,Manichaeon,%
2119   Manichaeon,Meroitic Cursive,Meroitic,Old North Arabian,%
2120   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
2121   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
2122   Old South Arabian,}%
2123 \def\bbl@provide@dirs#1{%
2124   \bbl@xin@\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
2125   \ifin@
2126     \global\bbl@csarg\chardef{wdir@#1}\@ne
2127     \bbl@xin@\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
2128     \ifin@
2129     \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
2130     \fi
2131   \else
2132     \global\bbl@csarg\chardef{wdir@#1}\z@
2133     \fi}
2134 \def\bbl@switchdir{%
2135   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{%}%
2136   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{%}%
2137   \bbl@exp{\bbl@setdirs\bbl@cs{wdir@\languagename}}}}
2138 \def\bbl@setdirs#1{% TODO - math
2139   \ifcase\bbl@select@type % TODO - strictly, not the right test
2140     \bbl@bodydir{#1}%
2141     \bbl@paddir{#1}%

```

```

2142 \fi
2143 \bbl@textdir{#1}}
2144 \ifodd\bbl@engine % luatex=1
2145 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2146 \DisableBabelHook{babel-bidi}
2147 \def\bbl@getluadir#1{%
2148   \directlua{
2149     if tex.#1dir == 'TLT' then
2150       tex.sprint('0')
2151     elseif tex.#1dir == 'TRT' then
2152       tex.sprint('1')
2153     end}}
2154 \def\bbl@setdir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
2155   \ifcase#3\relax
2156     \ifcase\bbl@getluadir{#1}\relax\else
2157       #2 TLT\relax
2158     \fi
2159   \else
2160     \ifcase\bbl@getluadir{#1}\relax
2161       #2 TRT\relax
2162     \fi
2163   \fi}
2164 \def\bbl@textdir#1{%
2165   \bbl@setdir{text}\textdir{#1}% TODO - ?\linedir
2166   \setattribute\bbl@attr@dir{#1}}
2167 \def\bbl@pardir{\bbl@setdir{par}\pardir}
2168 \def\bbl@bodydir{\bbl@setdir{body}\bodydir}
2169 \def\bbl@pagedir{\bbl@setdir{page}\pagedir}
2170 \def\bbl@dirparastext{\pardir\the\textdir\relax}% %%%
2171 \else % pdftex=0, xetex=2
2172 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2173 \DisableBabelHook{babel-bidi}
2174 \newcount\bbl@dirlevel
2175 \chardef\bbl@thetextdir\z@
2176 \chardef\bbl@thepardir\z@
2177 \def\bbl@textdir#1{%
2178   \ifcase#1\relax
2179     \chardef\bbl@thetextdir\z@
2180     \bbl@textdir@i\beginL\endL
2181   \else
2182     \chardef\bbl@thetextdir\@ne
2183     \bbl@textdir@i\beginR\endR
2184   \fi}
2185 \def\bbl@textdir@i#1#2{%
2186   \ifhmode
2187     \ifnum\currentgrouplevel>\z@
2188       \ifnum\currentgrouplevel=\bbl@dirlevel
2189         \bbl@error{Multiple bidi settings inside a group}%
2190           {I'll insert a new group, but expect wrong results.}%
2191         \bgroup\aftergroup#2\aftergroup\egroup
2192       \else
2193         \ifcase\currentgrouptype\or % 0 bottom
2194           \aftergroup#2% 1 simple {}
2195         \or
2196           \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
2197         \or
2198           \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
2199         \or\or\or % vbox vtop align
2200         \or

```

```

2201         \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
2202         \or\or\or\or\or\or % output math disc insert vcent mathchoice
2203         \or
2204         \aftergroup#2% 14 \beginngroup
2205         \else
2206         \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
2207         \fi
2208         \fi
2209         \bbl@dirlevel\currentgrouplevel
2210         \fi
2211         #1%
2212         \fi}
2213 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
2214 \let\bbl@bodydir@gobble
2215 \let\bbl@pagedir@gobble
2216 \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).

```

2217 \def\bbl@xebidipar{%
2218   \let\bbl@xebidipar\relax
2219   \TeXeTstate\@ne
2220   \def\bbl@xeverypar{%
2221     \ifcase\bbl@thepardir
2222       \ifcase\bbl@thetextdir\else\beginR\fi
2223     \else
2224       {\setbox\z@\lastbox\beginR\box\z@}%
2225     \fi}%
2226   \let\bbl@severypar\everypar
2227   \newtoks\everypar
2228   \everypar=\bbl@severypar
2229   \bbl@severypar{\bbl@xeverypar\the\everypar}}
2230 \fi

```

A tool for weak L (mainly digits).

```

2231 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}

```

## 9.8 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 `nor.sk.cfg` will be loaded when the language definition file `nor.sk.ldf` is loaded.

For plain-based formats we don't want to override the definition of `\loadlocalcfg` from `plain.def`.

```

2232 \ifx\loadlocalcfg\undefined
2233   \ifpackagewith{babel}{noconfigs}%
2234     {\let\loadlocalcfg@gobble}%
2235     {\def\loadlocalcfg#1{%
2236       \InputIfFileExists{#1.cfg}%
2237       {\typeout{*****^J%
2238                 * Local config file #1.cfg used^^J%
2239                 *}}%
2240       \@empty}}
2241 \fi

```

Just to be compatible with L<sup>A</sup>T<sub>E</sub>X 2.09 we add a few more lines of code:

```
2242 \ifx\@unexpandable@protect\@undefined
2243 \def\@unexpandable@protect{\noexpand\protect\noexpand}
2244 \long\def\protected@write#1#2#3{%
2245   \begingroup
2246     \let\thepage\relax
2247     #2%
2248     \let\protect\@unexpandable@protect
2249     \edef\reserved@a{\write#1{#3}}%
2250     \reserved@a
2251   \endgroup
2252   \if@nobreak\ifvmode\nobreak\fi\fi}
2253 \fi
2254 </core>
```

## 10 Multiple languages (switch.def)

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.

```
2255 <*kernel>
2256 <<Make sure ProvidesFile is defined>>
2257 \ProvidesFile{switch.def}[\<date>] [\<version>] Babel switching mechanism]
2258 <<Load macros for plain if not LaTeX>>
2259 <<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.

```
2260 \def\bb1@version{\<version>}
2261 \def\bb1@date{\<date>}
2262 \def\adddialect#1#2{%
2263   \global\chardef#1#2\relax
2264   \bb1@usehooks{adddialect}{#1}{#2}}%
2265   \log{\string#1 = a dialect from \string\language#2}}
```

`\bb1@iflanguage` executes code only if the language `l@` exists. Otherwise raises an error. The argument of `\bb1@fixname` has to be a macro name, as it may get “fixed” if casing (l/c) is wrong. It’s intended to fix a long-standing bug when `\foreignlanguage` and the like appear in a `\MakeXXXcase`. However, a lowercase form is not imposed to improve backward compatibility (perhaps you defined a language named MYLANG, but unfortunately mixed case names cannot be trapped). Note `l@` is encapsulated, so that its case does not change.

```
2266 \def\bb1@fixname#1{%
2267   \begingroup
2268     \def\bb1@tempe{l@}%
2269     \edef\bb1@tempd{\noexpand\@ifundefined{\noexpand\bb1@tempe#1}}%
2270     \bb1@tempd
2271     {\lowercase\expandafter{\bb1@tempd}}%
2272     {\uppercase\expandafter{\bb1@tempd}}%
2273     \@empty
2274     {\edef\bb1@tempd{\def\noexpand#1{#1}}%
2275      \uppercase\expandafter{\bb1@tempd}}}%
2276     {\edef\bb1@tempd{\def\noexpand#1{#1}}%
2277      \lowercase\expandafter{\bb1@tempd}}}%
2278   \@empty
```

```

2279 \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
2280 \bbl@tempd}
2281 \def\bbl@iflanguage#1{%
2282 \@ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}

```

`\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.

```

2283 \def\iflanguage#1{%
2284 \bbl@iflanguage{#1}{%
2285 \ifnum\csname l@#1\endcsname=\language
2286 \expandafter\@firstoftwo
2287 \else
2288 \expandafter\@secondoftwo
2289 \fi}}

```

## 10.1 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.

To allow the call of `\selectlanguage` either with a control sequence name or with a simple string as argument, we have to use a trick to delete the optional escape character. To convert a control sequence to a string, we use the `\string` primitive. Next we have to look at the first character of this string and compare it with the escape character. Because this escape character can be changed by setting the internal integer `\escapechar` to a character number, we have to compare this number with the character of the string. To do this we have to use TeX's backquote notation to specify the character as a number. If the first character of the `\string`'ed argument is the current escape character, the comparison has stripped this character and the rest in the 'then' part consists of the rest of the control sequence name. Otherwise we know that either the argument is not a control sequence or `\escapechar` is set to a value outside of the character range 0–255. If the user gives an empty argument, we provide a default argument for `\string`. This argument should expand to nothing.

```

2290 \let\bbl@select@type\z@
2291 \edef\selectlanguage{%
2292 \noexpand\protect
2293 \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`.

```

2294 \ifx\@undefined\protect\let\protect\relax\fi

```

As L<sup>A</sup>T<sub>E</sub>X 2.09 writes to files *expanded* whereas L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> takes care *not* to expand the arguments of `\write` statements we need to be a bit clever about the way we add information to .aux files. Therefore we introduce the macro `\xstring` which should expand to the right amount of `\string`'s.

```

2295 \ifx\documentclass\@undefined
2296 \def\xstring{\string\string\string}
2297 \else
2298 \let\xstring\string
2299 \fi

```

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.

```
2300 \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`

```
2301 \def\bbl@push@language{%
2302   \xdef\bbl@language@stack{\languagename+\bbl@language@stack}}
```

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

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

```
2303 \def\bbl@pop@lang#1+#2-#3{%
2304   \edef\languagename{#1}\xdef#3{#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) followed by the '-'-sign and finally the reference to the stack.

```
2305 \let\bbl@ifrestoring\@secondoftwo
2306 \def\bbl@pop@language{%
2307   \expandafter\bbl@pop@lang\bbl@language@stack-\bbl@language@stack
2308   \let\bbl@ifrestoring\@firstoftwo
2309   \expandafter\bbl@set@language\expandafter{\languagename}%
2310   \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.

```
2311 \expandafter\def\csname selectlanguage \endcsname#1{%
2312   \ifnum\bbl@hymapsel=\@ccclv\let\bbl@hymapsel\tw@fi
2313   \bbl@push@language
2314   \aftergroup\bbl@pop@language
2315   \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 not well defined. 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.

```

2316 \def\BabelContentsFiles{toc,lof,lot}
2317 \def\bbl@set@language#1{%
2318   \edef\language{%
2319     \ifnum\escapechar=\expandafter`\string#1\@empty
2320     \else\string#1\@empty\fi}%
2321   \select@language{\language}%
2322   \expandafter\ifx\csgname date\language\endcsname\relax\else
2323     \if@filesw
2324       \protected@write\@auxout{{}\string\babel@aux{\language}{}}%
2325       \bbl@usehooks{write}{}%
2326     \fi
2327   \fi}
2328 \def\select@language#1{%
2329   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
2330   \edef\language{#1}%
2331   \bbl@fixname\language
2332   \bbl@iflanguage\language{%
2333     \expandafter\ifx\csgname date\language\endcsname\relax
2334     \bbl@error
2335     {Unknown language `#1'. Either you have\\%
2336      misspelled its name, it has not been installed,\\%
2337      or you requested it in a previous run. Fix its name,\\%
2338      install it or just rerun the file, respectively}%
2339     {You may proceed, but expect wrong results}%
2340   \else
2341     \let\bbl@select@type\z@
2342     \expandafter\bbl@switch\expandafter{\language}%
2343     \fi}}
2344 \def\babel@aux#1#2{%
2345   \select@language{#1}%
2346   \bbl@foreach\BabelContentsFiles{%
2347     \@writefile{##1}{\babel@toc{#1}{#2}}} %% TODO - ok in plain?
2348 \def\babel@toc#1#2{%
2349   \select@language{#1}}

```

A bit of optimization. Select in heads/foots the language only if necessary. The real thing is in `babel.def`.

```

2350 \let\select@language@x\select@language

```

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 `\<lang>hyphenmins` is

defined. If it is not, we set default values (2 and 3), otherwise the values in `\(lang)hyphenmins` will be used.

```

2351 \newif\ifbbl@usedategroup
2352 \def\bbl@switch#1{%
2353   \originalTeX
2354   \expandafter\def\expandafter\originalTeX\expandafter{%
2355     \csname noextras#1\endcsname
2356     \let\originalTeX\@empty
2357     \babel@beginsave}%
2358   \bbl@usehooks{afterreset}}}%
2359   \languageshorthands{none}%
2360   \ifcase\bbl@select@type
2361     \ifhmode
2362       \hskip\z@skip % trick to ignore spaces
2363       \csname captions#1\endcsname\relax
2364       \csname date#1\endcsname\relax
2365       \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
2366     \else
2367       \csname captions#1\endcsname\relax
2368       \csname date#1\endcsname\relax
2369     \fi
2370   \else\ifbbl@usedategroup
2371     \bbl@usedategroupfalse
2372     \ifhmode
2373       \hskip\z@skip % trick to ignore spaces
2374       \csname date#1\endcsname\relax
2375       \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
2376     \else
2377       \csname date#1\endcsname\relax
2378     \fi
2379   \fi\fi
2380   \bbl@usehooks{beforeextras}}}%
2381   \csname extras#1\endcsname\relax
2382   \bbl@usehooks{afterextras}}}%
2383   \ifcase\bbl@opt@hyphenmap\or
2384     \def\BabelLower##1##2{\lccode##1=##2\relax}%
2385     \ifnum\bbl@hymapsel>4\else
2386       \csname\languagenam @bbl@hyphenmap\endcsname
2387     \fi
2388     \chardef\bbl@opt@hyphenmap\z@
2389   \else
2390     \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
2391       \csname\languagenam @bbl@hyphenmap\endcsname
2392     \fi
2393   \fi
2394   \global\let\bbl@hymapsel\@cclv
2395   \bbl@patterns{#1}%
2396   \babel@savevariable\lefthyphenmin
2397   \babel@savevariable\righthyphenmin
2398   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2399     \set@hyphenmins\tw@\thr@\relax
2400   \else
2401     \expandafter\expandafter\expandafter\set@hyphenmins
2402     \csname #1hyphenmins\endcsname\relax
2403   \fi}

```

other language The other language environment can be used as an alternative to using the `\selectlanguage` declarative command. When you are typesetting a document which

mixes left-to-right and right-to-left typesetting you have to use this environment in order to let things work as you expect them to.

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

```
2404 \long\def\otherlanguage#1{%
2405   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@\fi
2406   \csname selectlanguage \endcsname{#1}%
2407   \ignorespaces}
```

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

```
2408 \long\def\endotherlanguage{%
2409   \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`.

```
2410 \expandafter\def\csname otherlanguage*\endcsname#1{%
2411   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
2412   \foreign@language{#1}}
```

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”.

```
2413 \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 behaviour 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`.

```
2414 \let\bbl@beforeforeign\@empty
2415 \edef\foreignlanguage{%
2416   \noexpand\protect
2417   \expandafter\noexpand\csname foreignlanguage \endcsname}
2418 \expandafter\def\csname foreignlanguage \endcsname{%
2419   \@ifstar\bbl@foreign@s\bbl@foreign@x}
2420 \def\bbl@foreign@x#1#2{%
```

```

2421 \begingroup
2422   \let\BabelText\@firstofone
2423   \bbl@beforeforeign
2424   \foreign@language{#1}%
2425   \bbl@usehooks{foreign}{}%
2426   \BabelText{#2}% Now in horizontal mode!
2427 \endgroup}
2428 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
2429 \begingroup
2430   {\par}%
2431   \let\BabelText\@firstofone
2432   \foreign@language{#1}%
2433   \bbl@usehooks{foreign*}{}%
2434   \bbl@dirparastext
2435   \BabelText{#2}% Still in vertical mode!
2436   {\par}%
2437 \endgroup}

```

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

```

2438 \def\foreign@language#1{%
2439   \edef\languagename{#1}%
2440   \bbl@fixname\languagename
2441   \bbl@iflanguage\languagename{%
2442     \expandafter\ifx\csname date\languagename\endcsname\relax
2443       \bbl@warning
2444       {Unknown language `#1'. Either you have\\%
2445        misspelled its name, it has not been installed,\\%
2446        or you requested it in a previous run. Fix its name,\\%
2447        install it or just rerun the file, respectively.\\%
2448        I'll proceed, but expect wrong results.\\%
2449        Reported}%
2450     \fi
2451     \let\bbl@select@type\@ne
2452     \expandafter\bbl@switch\expandafter{\languagename}}

```

`\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.

```

2453 \let\bbl@hyphlist\@empty
2454 \let\bbl@hyphenation@\relax
2455 \let\bbl@pttnlist\@empty
2456 \let\bbl@patterns@\relax
2457 \let\bbl@hymapsel=\@ccclv
2458 \def\bbl@patterns#1{%
2459   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
2460     \csname l@#1\endcsname
2461     \edef\bbl@tempa{#1}%
2462     \else
2463     \csname l@#1:\f@encoding\endcsname

```

```

2464     \edef\bbl@tempa{#1:\f@encoding}%
2465     \fi
2466     \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}%
2467     \@ifundefined{bbl@hyphenation@}{% Can be \relax!
2468     \begingroup
2469     \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
2470     \ifin@else
2471     \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}%
2472     \hyphenation{%
2473     \bbl@hyphenation@
2474     \@ifundefined{bbl@hyphenation@#1}%
2475     \@empty
2476     {\space\csname bbl@hyphenation@#1\endcsname}}%
2477     \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
2478     \fi
2479     \endgroup}}

```

`hyphenrules` The environment `hyphenrules` can be used to select *just* the hyphenation rules. This environment does *not* change `\languagename` 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 `other language*`.

```

2480 \def\hyphenrules#1{%
2481   \edef\bbl@tempf{#1}%
2482   \bbl@fixname\bbl@tempf
2483   \bbl@iflanguage\bbl@tempf{%
2484     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
2485     \languageshortands{none}%
2486     \bbl@ifunset{\bbl@tempf hyphenmins}%
2487     {\set@hyphenmins\tw@\thr@\relax}%
2488     {\bbl@exp{\set@hyphenmins\@nameuse{\bbl@tempf hyphenmins}}}}
2489 \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.

```

2490 \def\providehyphenmins#1#2{%
2491   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2492   \@namedef{#1hyphenmins}{#2}%
2493   \fi}

```

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

```

2494 \def\set@hyphenmins#1#2{%
2495   \lefthyphenmin#1\relax
2496   \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.

```

2497 \ifx\ProvidesFile\@undefined
2498   \def\ProvidesLanguage#1[#2 #3 #4]{%
2499     \wlog{Language: #1 #4 #3 <#2>}%
2500     }
2501 \else
2502   \def\ProvidesLanguage#1{%
2503     \begingroup

```

```

2504     \catcode`\ 10 %
2505     \@makeother\/%
2506     \ifnextchar[%]
2507         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
2508     \def\@provideslanguage#1[#2]{%
2509         \wlog{Language: #1 #2}%
2510         \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
2511         \endgroup}
2512 \fi

```

`\LdfInit` This macro is defined in two versions. The first version is to be part of the ‘kernel’ of babel, ie. the part that is loaded in the format; the second version is defined in `babel.def`. The version in the format just checks the category code of the ampersand and then loads `babel.def`.

The category code of the ampersand is restored and the macro calls itself again with the new definition from `babel.def`

```

2513 \def\LdfInit{%
2514     \chardef\atcatcode=\catcode`\@
2515     \catcode`\@=11\relax
2516     \input babel.def\relax
2517     \catcode`\@=\atcatcode \let\atcatcode\relax
2518     \LdfInit}

```

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

```

2519 \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 initialises the save mechanism, `\babel@beginsave`, is not considered to be undefined.

```

2520 \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’:

```

2521 \providecommand\setlocale{%
2522     \bbl@error
2523     {Not yet available}%
2524     {Find an armchair, sit down and wait}}
2525 \let\uselocale\setlocale
2526 \let\locale\setlocale
2527 \let\selectlocale\setlocale
2528 \let\textlocale\setlocale
2529 \let\textlanguage\setlocale
2530 \let\languagetext\setlocale

```

## 10.2 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  $\LaTeX 2_{\epsilon}$ , so we can safely use its error handling interface. Otherwise we'll have to 'keep it simple'.

```

2531 \edef\bbl@nulllanguage{\string\language=0}
2532 \ifx\PackageError\@undefined
2533   \def\bbl@error#1#2{%
2534     \begingroup
2535       \newlinechar=`^^J
2536       \def\{^^J(babel) }%
2537       \errhelp{#2}\errmessage{\#1}%
2538     \endgroup}
2539   \def\bbl@warning#1{%
2540     \begingroup
2541       \newlinechar=`^^J
2542       \def\{^^J(babel) }%
2543       \message{\#1}%
2544     \endgroup}
2545   \def\bbl@info#1{%
2546     \begingroup
2547       \newlinechar=`^^J
2548       \def\{^^J}%
2549       \wlog{#1}%
2550     \endgroup}
2551 \else
2552   \def\bbl@error#1#2{%
2553     \begingroup
2554       \def\{\MessageBreak}%
2555       \PackageError{babel}{#1}{#2}%
2556     \endgroup}
2557   \def\bbl@warning#1{%
2558     \begingroup
2559       \def\{\MessageBreak}%
2560       \PackageWarning{babel}{#1}%
2561     \endgroup}
2562   \def\bbl@info#1{%
2563     \begingroup
2564       \def\{\MessageBreak}%
2565       \PackageInfo{babel}{#1}%
2566     \endgroup}
2567 \fi
2568 \@ifpackagewith{babel}{silent}
2569   {\let\bbl@info@gobble
2570    \let\bbl@warning@gobble}
2571   {}
2572 \def\bbl@nocaption#1#2{% 1: text to be printed 2: caption macro \langXname
2573   \gdef#2{\textbf{?#1?}}%
2574   #2%
2575   \bbl@warning{%
2576     \string#2 not set. Please, define\%
2577     it in the preamble with something like:\%
2578     \string\renewcommand\string#2{..\}%
2579     Reported}}
2580 \def\@nolanerr#1{%
2581   \bbl@error
2582   {You haven't defined the language #1\space yet}%
2583   {Your command will be ignored, type <return> to proceed}}
2584 \def\@nopatterns#1{%
2585   \bbl@warning
2586   {No hyphenation patterns were preloaded for\%

```

```

2587     the language `#1' into the format.\%
2588     Please, configure your TeX system to add them and\%
2589     rebuild the format. Now I will use the patterns\%
2590     preloaded for \bbl@nulllanguage\space instead}}
2591 \let\bbl@usehooks\@gobbletwo
2592 </kernel>

```

## 11 Loading hyphenation patterns

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

`toks8` stores info to be shown when the program is run.

We want to add a message to the message  $\text{\LaTeX}$  2.09 puts in the `\everyjob` register. This could be done by the following code:

```

\let\orgeveryjob\everyjob
\def\everyjob#1{%
  \orgeveryjob{#1}%
  \orgeveryjob\expandafter{\the\orgeveryjob\immediate\write16{%
    hyphenation patterns for \the\loaded@patterns loaded.}}%
  \let\everyjob\orgeveryjob\let\orgeveryjob\@undefined}

```

The code above redefines the control sequence `\everyjob` in order to be able to add something to the current contents of the register. This is necessary because the processing of hyphenation patterns happens long before  $\text{\LaTeX}$  fills the register.

There are some problems with this approach though.

- When someone wants to use several hyphenation patterns with  $\text{\SL\TeX}$  the above scheme won't work. The reason is that  $\text{\SL\TeX}$  overwrites the contents of the `\everyjob` register with its own message.
- Plain  $\text{\TeX}$  does not use the `\everyjob` register so the message would not be displayed.

To circumvent this a 'dirty trick' can be used. As this code is only processed when creating a new format file there is one command that is sure to be used, `\dump`. Therefore the original `\dump` is saved in `\orig@dump` and a new definition is supplied.

To make sure that  $\text{\LaTeX}$  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.

```

2593 (*patterns)
2594 <<Make sure ProvidesFile is defined>>
2595 \ProvidesFile{hyphen.cfg}[<<date>> <<version>> Babel hyphens]
2596 \xdef\bbl@format{\jobname}
2597 \ifx\AtBeginDocument\@undefined
2598   \def\@empty{}
2599   \let\orig@dump\dump
2600   \def\dump{%
2601     \ifx\@ztryfc\@undefined
2602     \else
2603       \toks0=\expandafter{\@preamblecmds}%

```

```

2604     \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
2605     \def\@begindocumenthook{%
2606     \fi
2607     \let\dump\orig@dump\let\orig@dump\undefined\dump}
2608 \fi
2609 <<Define core switching macros>>
2610 \toks8{Babel <<@version>> and hyphenation patterns for }%

```

`\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.

```

2611 \def\process@line#1#2 #3 #4 {%
2612 \ifx=#1%
2613 \process@synonym{#2}%
2614 \else
2615 \process@language{#1#2}{#3}{#4}%
2616 \fi
2617 \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.

```

2618 \toks@{}
2619 \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.

```

2620 \def\process@synonym#1{%
2621 \ifnum\last@language=\m@ne
2622 \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
2623 \else
2624 \expandafter\chardef\csname l@#1\endcsname\last@language
2625 \wlog{\string\l@#1=\string\language\the\last@language}%
2626 \expandafter\let\csname #1hyphenmins\expandafter\endcsname
2627 \csname\languagenamename hyphenmins\endcsname
2628 \let\bbl@elt\relax
2629 \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
2630 \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 ‘name’ of the language that will be loaded now is added to the token register `\toks8`, and finally 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 behaviour 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 `\langle lang \rangle 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.

```

2631 \def\process@language#1#2#3{%
2632   \expandafter\addlanguage\csname l@#1\endcsname
2633   \expandafter\language\csname l@#1\endcsname
2634   \edef\languagename{#1}%
2635   \bbl@hook@everylanguage{#1}%
2636   \bbl@get@enc#1::\@@@
2637   \begingroup
2638     \lefthyphenmin\m@ne
2639     \bbl@hook@loadpatterns{#2}%
2640     \ifnum\lefthyphenmin=\m@ne
2641     \else
2642       \expandafter\xdef\csname #1hyphenmins\endcsname{%
2643         \the\lefthyphenmin\the\righthyphenmin}%
2644     \fi
2645   \endgroup
2646   \def\bbl@tempa{#3}%
2647   \ifx\bbl@tempa\@empty\else
2648     \bbl@hook@loadexceptions{#3}%
2649   \fi
2650   \let\bbl@elt\relax
2651   \edef\bbl@languages{%
2652     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
2653   \ifnum\the\language=\z@
2654     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2655       \set@hyphenmins\tw@\thr@@\relax
2656     \else
2657       \expandafter\expandafter\expandafter\set@hyphenmins
2658         \csname #1hyphenmins\endcsname
2659     \fi
2660     \the\toks@
2661     \toks@{}%
2662   \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.

```

2663 \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.

```

2664 \def\bbl@hook@everylanguage#1{}
2665 \def\bbl@hook@loadpatterns#1{\input #1\relax}
2666 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
2667 \let\bbl@hook@loadkernel\bbl@hook@loadpatterns
2668 \begingroup
2669   \def\AddBabelHook#1#2{%
2670     \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
2671       \def\next{\toks1}%
2672     \else
2673       \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
2674     \fi
2675     \next}
2676 \if\directlua@undefined
2677   \if\XeTeXinputencoding@undefined\else
2678     \input xebabel.def
2679   \fi
2680 \else
2681   \input luababel.def
2682 \fi
2683 \openin1 = babel-\bbl@format.cfg
2684 \ifeof1
2685 \else
2686   \input babel-\bbl@format.cfg\relax
2687 \fi
2688 \closein1
2689 \endgroup
2690 \bbl@hook@loadkernel{switch.def}

```

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

```
2691 \openin1 = language.dat
```

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

```

2692 \def\languagename{english}%
2693 \ifeof1
2694   \message{I couldn't find the file language.dat,\space
2695           I will try the file hyphen.tex}
2696   \input hyphen.tex\relax
2697   \chardef\l@english\z@
2698 \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`.

```
2699 \last@language\m@ne
```

We now read lines from the file until the end is found

```
2700 \loop
```

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.

```

2701   \endlinechar\m@ne
2702   \read1 to \bbl@line
2703   \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.

```

2704 \if T\ifeof1F\fi T\relax
2705 \ifx\bbl@line\@empty\else
2706 \edef\bbl@line{\bbl@line\space\space\space}%
2707 \expandafter\process@line\bbl@line\relax
2708 \fi
2709 \repeat

```

Check for the end of the file. We must reverse the test for `\ifeof` without `\else`. Then reactivate the default patterns.

```

2710 \begingroup
2711 \def\bbl@elt#1#2#3#4{%
2712 \global\language=#2\relax
2713 \gdef\languagename{#1}%
2714 \def\bbl@elt##1##2##3##4{}}%
2715 \bbl@languages
2716 \endgroup
2717 \fi

```

and close the configuration file.

```
2718 \closein1
```

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

```

2719 \if\the\toks@\else
2720 \errhelp{language.dat loads no language, only synonyms}
2721 \errmessage{Orphan language synonym}
2722 \fi
2723 \advance\last@language\@ne
2724 \edef\bbl@tempa{%
2725 \everyjob{%
2726 \the\everyjob
2727 \ifx\typeout\@undefined
2728 \immediate\write16%
2729 \else
2730 \noexpand\typeout
2731 \fi
2732 {\the\toks8 \the\last@language\space language(s) loaded.}}}
2733 \advance\last@language\m@ne
2734 \bbl@tempa

```

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.

```

2735 \let\bbl@line\@undefined
2736 \let\process@line\@undefined
2737 \let\process@synonym\@undefined
2738 \let\process@language\@undefined
2739 \let\bbl@get@enc\@undefined
2740 \let\bbl@hyph@enc\@undefined
2741 \let\bbl@tempa\@undefined
2742 \let\bbl@hook@loadkernel\@undefined
2743 \let\bbl@hook@everylanguage\@undefined
2744 \let\bbl@hook@loadpatterns\@undefined
2745 \let\bbl@hook@loadexceptions\@undefined
2746 </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].

```
2747 <<{*More package options}>> ≡
2748 \iffodd\bbl@engine
2749 \DeclareOption{bidi=basic-r}%
2750   {\let\bbl@beforeforeign\leavevmode
2751     \newattribute\bbl@attr@dir
2752     \bbl@exp{\output{\bodydir\pagedir\the\output}}}%
2753   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
2754 \else
2755   \DeclareOption{bidi=basic-r}%
2756     {\bbl@error
2757       {The bidi method `basic-r' is available only in\\%
2758         luatex. I'll continue with `bidi=default', so\\%
2759         expect wrong results}%
2760       {See the manual for further details.}%
2761     \let\bbl@beforeforeign\leavevmode
2762     \AtEndOfPackage{%
2763       \EnableBabelHook{babel-bidi}%
2764       \bbl@xebidipar}}
2765 \fi
2766 \DeclareOption{bidi=default}%
2767   {\let\bbl@beforeforeign\leavevmode
2768     \iffodd\bbl@engine
2769       \newattribute\bbl@attr@dir
2770       \bbl@exp{\output{\bodydir\pagedir\the\output}}}%
2771   \fi
2772   \AtEndOfPackage{%
2773     \EnableBabelHook{babel-bidi}%
2774     \iffodd\bbl@engine\else
2775       \bbl@xebidipar
2776     \fi}}
2777 <</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.

```
2778 <<{*Font selection}>> ≡
2779 \@onlypreamble\babelfont
2780 \newcommand\babelfont[2][]{% 1=langs/scripts 2=fam
2781   \edef\bbl@tempa{#1}%
2782   \def\bbl@tempb{#2}%
2783   \ifx\fontspec@undefined
2784     \usepackage{fontspec}%
2785   \fi
2786   \EnableBabelHook{babel-fontspec}%
2787   \bbl@bblfont}
2788 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname
2789   \bbl@ifunset{\bbl@tempb family}{\bbl@providefam{\bbl@tempb}}}%
2790   \bbl@ifunset{\bbl@lsys@languagename}{\bbl@provide@lsys{\languagename}}}%
2791   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
2792   {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
2793   \bbl@exp{%
2794     \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
2795     \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
2796     \<bbl@tempb default>\<bbl@tempb family>}}%
```

```

2797   {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
2798     \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:

```

2799 \def\bbl@providfam#1{%
2800   \bbl@exp{%
2801     \\newcommand\<#1default>{% Just define it
2802     \\bbl@add@list\\bbl@font@fams{#1}%
2803     \\DeclareRobustCommand\<#1family>{%
2804       \\not@math@alphabet\<#1family>\relax
2805       \\fontfamily\<#1default>\\selectfont}%
2806     \\DeclareTextFontCommand{\<text#1>}{\<#1family>}}

```

The following macro is activated when the hook babel-fontspec is enabled.

```

2807 \def\bbl@switchfont{%
2808   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{%
2809     \bbl@exp{% eg Arabic -> arabic
2810       \lowercase{\def\\bbl@tempa{\bbl@cs{sname@\languagename}}}}%
2811   \bbl@foreach\bbl@font@fams{%
2812     \bbl@ifunset{bbl@##1dflt@\languagename}% (1) language?
2813     {\bbl@ifunset{bbl@##1dflt*@\bbl@tempa}% (2) from script?
2814       {\bbl@ifunset{bbl@##1dflt@}% 2=F - (3) from generic?
2815         {}% 123=F - nothing!
2816         {\bbl@exp{% 3=T - from generic
2817           \global\let\<bbl@##1dflt@\languagename>%
2818             \<bbl@##1dflt@>}}}%
2819         {\bbl@exp{% 2=T - from script
2820           \global\let\<bbl@##1dflt@\languagename>%
2821             \<bbl@##1dflt*@\bbl@tempa>}}}}%
2822     {}}% 1=T - language, already defined
2823 \def\bbl@tempa{%
2824   \bbl@warning{The current font is not a standard family.\\
2825     Script and Language are not applied. Consider defining\\
2826     a new family with \string\babelfont,}}%
2827 \bbl@foreach\bbl@font@fams{% don't gather with prev for
2828   \bbl@ifunset{bbl@##1dflt@\languagename}%
2829     {\bbl@cs{famrst@##1}%
2830     \global\bbl@csarg\let{famrst@##1}\relax}%
2831     {\bbl@exp{% order is relevant
2832       \\bbl@add\\originalTeX{%
2833         \\bbl@font@rst{\bbl@cs{##1dflt@\languagename}}%
2834           \<##1default>\<##1family>{##1}}%
2835         \\bbl@font@set\<bbl@##1dflt@\languagename>% the main part!
2836           \<##1default>\<##1family>}}}%
2837   \bbl@ifrestoring{ }\bbl@tempa}}%

```

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.

```

2838 \def\bbl@font@set#1#2#3{%
2839   \bbl@xin@{<>}{#1}%
2840   \ifin@
2841     \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1}%
2842   \fi
2843   \bbl@exp{%
2844     \def\\#2{#1}% eg, \rmddefault{\bbl@rm1dflt@lang}
2845     \\bbl@ifsamestring{#2}{\f@family}{\#3\let\\bbl@tempa\relax}}}}
2846 \def\bbl@fontspec@set#1#2#3{%
2847   \bbl@exp{\<fontspec_set_family:Nnn>\\#1%

```

```
2848   {\bbl@cs{lsys@\languagename},#2}}{#3}%
2849   \bbl@toglobal#1}%
```

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.

```
2850 \def\bbl@font@rst#1#2#3#4{%
2851   \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.

```
2852 \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 :-).

```
2853 \newcommand\babelFSstore[2][]{%
2854   \bbl@ifblank{#1}%
2855   {\bbl@csarg\def{sname@#2}{Latin}}%
2856   {\bbl@csarg\def{sname@#2}{#1}}%
2857   \bbl@provide@dirs{#2}%
2858   \bbl@csarg\ifnum{wdir@#2}>\z@
2859     \let\bbl@beforeforeign\leavevmode
2860     \EnableBabelHook{babel-bidi}%
2861   \fi
2862   \bbl@foreach{#2}{%
2863     \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
2864     \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
2865     \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
2866 \def\bbl@FSstore#1#2#3#4{%
2867   \bbl@csarg\edef{#2default#1}{#3}%
2868   \expandafter\addto\csname extras#1\endcsname{%
2869     \let#4#3%
2870     \ifx#3\f@family
2871       \edef#3{\csname bbl@#2default#1\endcsname}%
2872       \fontfamily{#3}\selectfont
2873     \else
2874       \edef#3{\csname bbl@#2default#1\endcsname}%
2875       \fi}%
2876   \expandafter\addto\csname noextras#1\endcsname{%
2877     \ifx#3\f@family
2878       \fontfamily{#4}\selectfont
2879     \fi
2880     \let#3#4}}
2881 \let\bbl@langfeatures\@empty
2882 \def\babelFSfeatures{% make sure \fontspec is redefined once
2883   \let\bbl@ori@fontspec\fontspec
2884   \renewcommand\fontspec[1][]{%
2885     \bbl@ori@fontspec[\bbl@langfeatures##1]}
2886   \let\babelFSfeatures\bbl@FSfeatures
2887   \babelFSfeatures}
2888 \def\bbl@FSfeatures#1#2{%
2889   \expandafter\addto\csname extras#1\endcsname{%
2890     \babel@save\bbl@langfeatures
2891     \edef\bbl@langfeatures{#2,}}
2892 <</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.

L<sup>A</sup>T<sub>E</sub>X sets many “codes” just before loading `hyphen.cfg`. That is not a problem in `luatex`, but in `xetex` they must be reset to the proper value. Most of the work is done in `xe(la)tex.ini`, so here we just “undo” some of the changes done by L<sup>A</sup>T<sub>E</sub>X. Anyway, for consistency `LuaTEX` also resets the catcodes.

```
2893 <<{*Restore Unicode catcodes before loading patterns}>> ≡
2894 \begingroup
2895 % Reset chars "80-"C0 to category "other", no case mapping:
2896 \catcode`\@=11 \count@=128
2897 \loop\ifnum\count@<192
2898 \global\uccode\count@=0 \global\lccode\count@=0
2899 \global\catcode\count@=12 \global\sffcode\count@=1000
2900 \advance\count@ by 1 \repeat
2901 % Other:
2902 \def\O ##1 {%
2903 \global\uccode"##1=0 \global\lccode"##1=0
2904 \global\catcode"##1=12 \global\sffcode"##1=1000 }%
2905 % Letter:
2906 \def\L ##1 ##2 ##3 {\global\catcode"##1=11
2907 \global\uccode"##1="##2
2908 \global\lccode"##1="##3
2909 % Uppercase letters have sffcode=999:
2910 \ifnum"##1="##3 \else \global\sffcode"##1=999 \fi }%
2911 % Letter without case mappings:
2912 \def\l ##1 {\L ##1 ##1 ##1 }%
2913 \l 00AA
2914 \L 00B5 039C 00B5
2915 \l 00BA
2916 \O 00D7
2917 \l 00DF
2918 \O 00F7
2919 \L 00FF 0178 00FF
2920 \endgroup
2921 \input #1\relax
2922 <</Restore Unicode catcodes before loading patterns>>
```

Now, the code.

```
2923 <{*xetex}>
2924 \def\BabelStringsDefault{unicode}
2925 \let\xebbl@stop\relax
2926 \AddBabelHook{xetex}{encodedcommands}{%
2927 \def\bbl@tempa{#1}%
2928 \ifx\bbl@tempa\@empty
2929 \XeTeXinputencoding"bytes"%
2930 \else
2931 \XeTeXinputencoding"#1"%
2932 \fi
2933 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
2934 \AddBabelHook{xetex}{stopcommands}{%
2935 \xebbl@stop
2936 \let\xebbl@stop\relax}
2937 \AddBabelHook{xetex}{loadkernel}{%
2938 <<Restore Unicode catcodes before loading patterns>>}
```

```

2939 \ifx\DisableBabelHook\@undefined\endinput\fi
2940 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
2941 \DisableBabelHook{babel-fontspec}
2942 <<Font selection>>
2943 \input txtbabel.def
2944 </xetex>

```

## 13.2 Layout

In progress.

Unfortunately, for proper support for xetex lots of macros and packages must be patched somehow. At least at this stage, babel will not do it and therefore a package similar to bidi will be required. Any help in making babel and bidi collaborate will be welcome. Note as well, 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  $\TeX$  expansion mechanism the following constructs are valid: \adim\bbl@startskip, \advance\bbl@startskip\adim, \bbl@startskip\adim.

Consider txtbabel as a shorthand for *tex-xet babel*, which the bidi model in both pdftex and xetex.

```

2945 <*texxet>
2946 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
2947 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
2948 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
2949 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
2950 \def\@hangfrom#1{%
2951 \setbox\@tempboxa\hbox{#1}%
2952 \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
2953 \noindent\box\@tempboxa}
2954 \def\raggedright{%
2955 \let\@centercr
2956 \bbl@startskip\z@skip
2957 \@rightskip\@flushglue
2958 \bbl@endskip\@rightskip
2959 \parindent\z@
2960 \parfillskip\bbl@startskip}
2961 \def\raggedleft{%
2962 \let\@centercr
2963 \bbl@startskip\@flushglue
2964 \bbl@endskip\z@skip
2965 \parindent\z@
2966 \parfillskip\bbl@endskip}
2967 \fi
2968 \IfBabelLayout{lists}
2969 {\def\list#1#2{%
2970 \ifnum \@listdepth >5\relax
2971 \@toodeep
2972 \else
2973 \global\advance\@listdepth\@ne
2974 \fi
2975 \rightmargin\z@
2976 \listparindent\z@
2977 \itemindent\z@
2978 \csname @list\romannumeral\the\@listdepth\endcsname
2979 \def\@itemlabel{#1}%
2980 \let\makelabel\@mklab
2981 \@nbrlistfalse

```

```

2982 #2\relax
2983 \@trivlist
2984 \parskip\parsep
2985 \parindent\listparindent
2986 \advance\linewidth-\rightmargin
2987 \advance\linewidth-\leftmargin
2988 \advance\@totalleftmargin
2989 \ifcase\@bbl@theparidir\leftmargin\else\rightmargin\fi
2990 \parshape\@ne\@totalleftmargin\linewidth
2991 \ignorespaces}%
2992 \def\labelenumii{}\theenumii}%
2993 \def\p@enumiii{\p@enumii}\theenumii{}}
2994 {}
2995 \IfBabelLayout{contents}
2996 {\def\@dottedtocline#1#2#3#4#5{%
2997 \ifnum#1>\c@tocdepth\else
2998 \vskip \z@ \@plus.2\p@
2999 {\bbl@startskip#2\relax
3000 \bbl@endskip\@tocmarg
3001 \parfillskip-\bbl@endskip
3002 \parindent#2\relax
3003 \@afterindenttrue
3004 \interlinepenalty\@M
3005 \leavevmode
3006 \@tempdima#3\relax
3007 \advance\bbl@startskip\@tempdima
3008 \null\nobreak\hskip-\bbl@startskip
3009 {#4}\nobreak
3010 \leaders\hbox{%
3011 $m@th\mkern\@dotsep mu\hbox{.}\mkern\@dotsep mu$}%
3012 \hfill\nobreak
3013 \hb@xt@\@pnumwidth{\hfil\normalfont\normalcolor#5}%
3014 \par}%
3015 \fi}}
3016 {}
3017 \IfBabelLayout{columns}
3018 {\def\@outputdblcol{%
3019 \if@firstcolumn
3020 \global\@firstcolumnfalse
3021 \global\setbox\@leftcolumn\copy\@outputbox
3022 \splitmaxdepth\maxdimen
3023 \vbadness\maxdimen
3024 \setbox\@outputbox\vbox{\unvbox\@outputbox\unskip}%
3025 \setbox\@outputbox\vsplit\@outputbox to\maxdimen
3026 \toks@ \expandafter{\topmark}%
3027 \xdef\@firstcoltopmark{\the\toks@}%
3028 \toks@ \expandafter{\splitfirstmark}%
3029 \xdef\@firstcolfirstmark{\the\toks@}%
3030 \ifx\@firstcolfirstmark\@empty
3031 \global\let\@setmarks\relax
3032 \else
3033 \gdef\@setmarks{%
3034 \let\firstmark\@firstcolfirstmark
3035 \let\topmark\@firstcoltopmark}%
3036 \fi
3037 \else
3038 \global\@firstcolumntrue
3039 \setbox\@outputbox\vbox{%
3040 \hb@xt@\textwidth{%

```

```

3041         \hskip\columnwidth
3042         \hfil
3043         {\normalcolor\vrule \@width\columnseprule}%
3044         \hfil
3045         \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
3046         \hskip-\textwidth
3047         \hb@xt@\columnwidth{\box\@outputbox \hss}%
3048         \hskip\columnsep
3049         \hskip\columnwidth}}%
3050     \@combinedblfloats
3051     \@setmarks
3052     \@outputpage
3053     \begingroup
3054         \@dblfloatplacement
3055         \@startdblcolumn
3056         \@whilesw\if@fcolmade \fi{\@outputpage
3057         \@startdblcolumn}%
3058     \endgroup
3059 \fi}}%
3060 {}

```

Implicitly reverses sectioning labels in bidi=basic-r, because the full stop is not in contact with L numbers any more. A think there must be a better way.

```

3061 \IfBabelLayout{counters}% Global or language dependent? At begin doc?
3062 {\let\bbl@latinarabic=\@arabic
3063  \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
3064  \let\bbl@asciroman=\@roman
3065  \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
3066  \let\bbl@asciiRoman=\@Roman
3067  \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}{}}
3068 \texpet)

```

### 13.3 LuaTeX

The new 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 has 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`). 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.

```

3069 (*luatex)
3070 \ifx\AddBabelHook\@undefined
3071 \begingroup
3072 \toks@{}
3073 \count@ \z@ % 0=start, 1=0th, 2=normal
3074 \def\bbl@process@line#1#2 #3 #4 {%
3075   \ifx=#1%
3076     \bbl@process@synonym{#2}%
3077   \else
3078     \bbl@process@language{#1#2}{#3}{#4}%
3079   \fi
3080   \ignorespaces}
3081 \def\bbl@manylang{%
3082   \ifnum\bbl@last>\@ne
3083     \bbl@info{Non-standard hyphenation setup}%
3084   \fi
3085   \let\bbl@manylang\relax}
3086 \def\bbl@process@language#1#2#3{%
3087   \ifcase\count@
3088     \@ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
3089   \or
3090     \count@\tw@
3091   \fi
3092   \ifnum\count@=\tw@
3093     \expandafter\addlanguage\csname l@#1\endcsname
3094     \language\allocationnumber
3095     \chardef\bbl@last\allocationnumber
3096     \bbl@manylang
3097     \let\bbl@elt\relax
3098     \xdef\bbl@languages{%
3099       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
3100   \fi
3101   \the\toks@
3102   \toks@{}}
3103 \def\bbl@process@synonym@aux#1#2{%
3104   \global\expandafter\chardef\csname l@#1\endcsname#2\relax
3105   \let\bbl@elt\relax
3106   \xdef\bbl@languages{%
3107     \bbl@languages\bbl@elt{#1}{#2}{}}}%
3108 \def\bbl@process@synonym#1{%
3109   \ifcase\count@
3110     \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
3111   \or
3112     \@ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}{%
3113   \else
3114     \bbl@process@synonym@aux{#1}{\the\bbl@last}%
3115   \fi}
3116 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
3117   \chardef\l@english\z@
3118   \chardef\l@USenglish\z@

```

```

3119 \chardef\bbl@last\z@
3120 \global\@namedef{bbl@hyphendata@0}{\hyphen.tex}{}
3121 \gdef\bbl@languages{%
3122   \bbl@elt{english}{0}{\hyphen.tex}{}%
3123   \bbl@elt{USenglish}{0}{}{}}
3124 \else
3125   \global\let\bbl@languages@format\bbl@languages
3126   \def\bbl@elt#1#2#3#4{% Remove all except language 0
3127     \ifnum#2>\z@\else
3128       \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
3129     \fi}%
3130   \xdef\bbl@languages{\bbl@languages}%
3131 \fi
3132 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{} } % Define flags
3133 \bbl@languages
3134 \openin1=language.dat
3135 \ifeof1
3136   \bbl@warning{I couldn't find language.dat. No additional\\%
3137     patterns loaded. Reported}%
3138 \else
3139   \loop
3140     \endlinechar\m@ne
3141     \read1 to \bbl@line
3142     \endlinechar`\^^M
3143     \if T\ifeof1F\fi T\relax
3144     \ifx\bbl@line\@empty\else
3145       \edef\bbl@line{\bbl@line\space\space\space}%
3146       \expandafter\bbl@process@line\bbl@line\relax
3147     \fi
3148   \repeat
3149 \fi
3150 \endgroup
3151 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
3152 \ifx\babelcatcodetablenum\undefined
3153   \def\babelcatcodetablenum{5211}
3154 \fi
3155 \def\bbl@luapatterns#1#2{%
3156   \bbl@get@enc#1::\@@@
3157   \setbox\z@\hbox\bgroup
3158     \begingroup
3159       \ifx\catcodetable\undefined
3160         \let\savecatcodetable\luatexsavecatcodetable
3161         \let\initcatcodetable\luatexinitcatcodetable
3162         \let\catcodetable\luatexcatcodetable
3163       \fi
3164       \savecatcodetable\babelcatcodetablenum\relax
3165       \initcatcodetable\numexpr\babelcatcodetablenum+1\relax
3166       \catcodetable\numexpr\babelcatcodetablenum+1\relax
3167       \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
3168       \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~ =13
3169       \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
3170       \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
3171       \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
3172       \catcode`\`=12 \catcode`\'=12 \catcode`\`=12
3173       \input #1\relax
3174       \catcodetable\babelcatcodetablenum\relax
3175     \endgroup
3176   \def\bbl@tempa{#2}%
3177   \ifx\bbl@tempa\@empty\else

```

```

3178     \input #2\relax
3179     \fi
3180 \egroup}%
3181 \def\bbl@patterns@lua#1{%
3182   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
3183     \csname l@#1\endcsname
3184     \edef\bbl@tempa{#1}%
3185   \else
3186     \csname l@#1:\f@encoding\endcsname
3187     \edef\bbl@tempa{#1:\f@encoding}%
3188   \fi\relax
3189   \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
3190   \@ifundefined{bbl@hyphendata@the\language}%
3191     {\def\bbl@elt##1##2##3##4{%
3192       \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
3193         \def\bbl@tempb{##3}%
3194         \ifx\bbl@tempb@empty\else % if not a synonymous
3195           \def\bbl@tempc{##3}{##4}%
3196         \fi
3197         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
3198       \fi}%
3199     \bbl@languages
3200     \@ifundefined{bbl@hyphendata@the\language}%
3201       {\bbl@info{No hyphenation patterns were set for\%
3202         language ‘\bbl@tempa’. Reported}}%
3203       {\expandafter\expandafter\expandafter\bbl@luapatterns
3204         \csname bbl@hyphendata@the\language\endcsname}}}}
3205 \endinput\fi
3206 \begingroup
3207 \catcode`\%=12
3208 \catcode`\'=12
3209 \catcode`\ "=12
3210 \catcode`\:=12
3211 \directlua{
3212   Babel = Babel or {}
3213   function Babel.bytes(line)
3214     return line:gsub(".",
3215       function (chr) return unicode.utf8.char(string.byte(chr)) end)
3216   end
3217   function Babel.begin_process_input()
3218     if luatexbase and luatexbase.add_to_callback then
3219       luatexbase.add_to_callback('process_input_buffer',
3220         Babel.bytes, 'Babel.bytes')
3221     else
3222       Babel.callback = callback.find('process_input_buffer')
3223       callback.register('process_input_buffer', Babel.bytes)
3224     end
3225   end
3226   function Babel.end_process_input ()
3227     if luatexbase and luatexbase.remove_from_callback then
3228       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
3229     else
3230       callback.register('process_input_buffer', Babel.callback)
3231     end
3232   end
3233   function Babel.addpatterns(pp, lg)
3234     local lg = lang.new(lg)
3235     local pats = lang.patterns(lg) or ''
3236     lang.clear_patterns(lg)

```

```

3237 for p in pp:gmatch('[^%s]+') do
3238   ss = ''
3239   for i in string.utfcharacters(p:gsub('%d', '')) do
3240     ss = ss .. '%d?' .. i
3241   end
3242   ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
3243   ss = ss:gsub('%.%%d%?$', '%%.')
3244   pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
3245   if n == 0 then
3246     tex.sprint(
3247       [[\string\csname\space bbl@info\endcsname{New pattern: }
3248       .. p .. [{}]])
3249     pats = pats .. ' ' .. p
3250   else
3251     tex.sprint(
3252       [[\string\csname\space bbl@info\endcsname{Renew pattern: }
3253       .. p .. [{}]])
3254   end
3255 end
3256 lang.patterns(lg, pats)
3257 end
3258 }
3259 \endgroup
3260 \def\BabelStringsDefault{unicode}
3261 \let\luabbl@stop\relax
3262 \AddBabelHook{luatex}{encodedcommands}{%
3263   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
3264   \ifx\bbl@tempa\bbl@tempb\else
3265     \directlua{Babel.begin_process_input()}%
3266     \def\luabbl@stop{%
3267       \directlua{Babel.end_process_input()}%
3268     \fi}%
3269 \AddBabelHook{luatex}{stopcommands}{%
3270   \luabbl@stop
3271   \let\luabbl@stop\relax}
3272 \AddBabelHook{luatex}{patterns}{%
3273   \@ifundefined{bbl@hyphendata@the\language}%
3274     {\def\bbl@elt##1##2##3##4{%
3275       \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
3276       \def\bbl@tempb{##3}%
3277       \ifx\bbl@tempb\@empty\else % if not a synonymous
3278         \def\bbl@tempc{##3}{##4}%
3279       \fi
3280       \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
3281     \fi}%
3282   \bbl@languages
3283   \@ifundefined{bbl@hyphendata@the\language}%
3284     {\bbl@info{No hyphenation patterns were set for\%
3285       language '#2'. Reported}}%
3286     {\expandafter\expandafter\expandafter\bbl@luapatterns
3287       \csname bbl@hyphendata@the\language\endcsname}}}%
3288 \@ifundefined{bbl@patterns@}{}%
3289   \begingroup
3290     \bbl@xin@{, \number\language,}{, \bbl@pttnlist}%
3291   \ifin\else
3292     \ifx\bbl@patterns@\@empty\else
3293       \directlua{ Babel.addpatterns(
3294         [[\bbl@patterns@]], \number\language) }%
3295     \fi

```

```

3296     \ifundefined{bbl@patterns@#1}%
3297     \empty
3298     {\directlua{ Babel.addpatterns(
3299         [[\space\csname bbl@patterns@#1\endcsname]],
3300         \number\language) }}%
3301     \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
3302     \fi
3303     \endgroup}}
3304 \AddBabelHook{luatex}{everylanguage}{%
3305   \def\process@language##1##2##3{%
3306     \def\process@line#####1####2 #####3 #####4 {}}
3307 \AddBabelHook{luatex}{loadpatterns}{%
3308   \input #1\relax
3309   \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
3310     {#1}}
3311 \AddBabelHook{luatex}{loadexceptions}{%
3312   \input #1\relax
3313   \def\bbl@tempb##1##2{##1}{##1}%
3314   \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
3315     {\expandafter\expandafter\expandafter\bbl@tempb
3316       \csname bbl@hyphendata@the\language\endcsname}}

```

`\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.

```

3317 \onlypreamble\babelpatterns
3318 \AtEndOfPackage{%
3319   \newcommand\babelpatterns[2][\empty]{%
3320     \ifx\bbl@patterns@relax
3321       \let\bbl@patterns@\empty
3322     \fi
3323     \ifx\bbl@pttnlist\empty\else
3324       \bbl@warning{%
3325         You must not intermingle \string\selectlanguage\space and\%
3326         \string\babelpatterns\space or some patterns will not\%
3327         be taken into account. Reported}%
3328     \fi
3329     \ifx\empty#1%
3330       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
3331     \else
3332       \edef\bbl@tempb{\zap@space#1 \empty}%
3333       \bbl@for\bbl@tempa\bbl@tempb{%
3334         \bbl@fixname\bbl@tempa
3335         \bbl@iflanguage\bbl@tempa{%
3336           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
3337             \ifundefined{bbl@patterns@\bbl@tempa}%
3338               \empty
3339               {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
3340             #2}}%
3341     \fi}}

```

#### Common stuff.

```

3342 \AddBabelHook{luatex}{loadkernel}{%
3343   <<Restore Unicode catcodes before loading patterns>>}
3344 \ifx\DisableBabelHook\undefined\endinput\fi
3345 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
3346 \DisableBabelHook{babel-fontspec}
3347 <<Font selection>>

```

## 13.4 Layout

### Work in progress.

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.) and with `bidi=basic-r`, 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`.

```
3348 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
3349 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
3350   \def\@hangfrom#1{%
3351     \setbox\@tempboxa\hbox{{#1}}%
3352     \hangindent\wd\@tempboxa
3353     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
3354       \shapemode\@ne
3355     \fi
3356     \noindent\box\@tempboxa}
3357 \fi
3358 \IfBabelLayout{lists}
3359   {\def\list#1#2{%
3360     \ifnum \@listdepth >5\relax
3361       \@toodeep
3362     \else
3363       \global\advance\@listdepth\@ne
3364     \fi
3365     \rightmargin\z@
3366     \listparindent\z@
3367     \itemindent\z@
3368     \csname @list\romannumeral\the\@listdepth\endcsname
3369     \def\@itemlabel{#1}%
3370     \let\makelabel\@mklab
3371     \@nmbrrlistfalse
3372     #2\relax
3373     \@trivlist
3374     \parskip\parsep
3375     \parindent\listparindent
3376     \advance\linewidth -\rightmargin
3377     \advance\linewidth -\leftmargin
3378     \advance\totalleftmargin \leftmargin
3379     \parshape \@ne
3380     \@totalleftmargin \linewidth
3381     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
3382       \shapemode\tw@
3383     \fi
3384     \ignorespaces}}
3385   {}
```

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-r`, but there are some additional readjustments for `bidi=default`.

```
3386 \IfBabelLayout{counters}% Global or language dependent? At begin doc?
3387   {\let\bbl@latinarabic=\@arabic
3388     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
3389     \ifpackagewith{babel}{bidi=default}
3390       {\let\bbl@asciroman=\@roman
```

```

3391 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
3392 \let\bbl@asciiRoman=\@Roman
3393 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
3394 \def\labelenumii{\theenumii}%
3395 \def\p@enumiii{\p@enumii}\theenumii}}{}
3396 </luatex>

```

### 13.5 Auto bidi with basic-r

The file `babel-bidi.lua` currently only contains data. It is a large and boring file and it's not shown here. See the generated file.

Now the `basic-r` bidi mode. One of the aims is to implement a fast and simple bidi algorithm, with a single loop. I managed to do it for R texts, with a second smaller loop for a special case. The code is still somewhat chaotic, but its behavior is essentially correct. I cannot resist copying the following text from Emacs `bidi.c` (which also attempts to implement the bidi algorithm with a single loop):

Arrrrgh!! The UAX#9 algorithm is too deeply entrenched in the assumption of batch-style processing [...]. May the fleas of a thousand camels infest the armpits of those who design supposedly general-purpose algorithms by looking at their own implementations, and fail to consider other possible implementations!

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, *what* they do and *why*, and not only *how*), but I think (or I hope) I've managed to understand them.

In some sense, there are two bidi modes, one for numbers, and the other for text.

Furthermore, setting just the direction in R text is not enough, because there are actually *two* R modes (set explicitly in Unicode with RLM and ALM). In `babel` the `dir` is set by a higher protocol based on the language/script, which in turn sets the correct `dir` (<l>, <r> or <al>).

From UAX#9: "Where available, markup should be used instead of the explicit formatting characters". So, this simple version just ignores formatting characters. Actually, most of that annex is devoted to how to handle them.

BD14-BD16 are not implemented. Unicode (and the W3C) are making a great effort to deal with some special problematic cases in "streamed" plain text. I don't think this is the way to go – particular issues should be fixed by a high level interface taking into account the needs of the document. And here is where `luatex` excels, because everything related to bidi writing is under our control.

```

3397 (*basic-r)
3398 Babel = Babel or {}
3399
3400 require('babel-bidi.lua')
3401
3402 local characters = Babel.characters
3403 local ranges = Babel.ranges
3404
3405 local DIR = node.id("dir")
3406
3407 local function dir_mark(head, from, to, outer)
3408   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
3409   local d = node.new(DIR)
3410   d.dir = '+' .. dir
3411   node.insert_before(head, from, d)
3412   d = node.new(DIR)
3413   d.dir = '-' .. dir
3414   node.insert_after(head, to, d)

```

```

3415 end
3416
3417 function Babel.pre_otfload(head)
3418   local first_n, last_n           -- first and last char with nums
3419   local last_es                   -- an auxiliary 'last' used with nums
3420   local first_d, last_d           -- first and last char in L/R block
3421   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):

```

3422   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
3423   local strong_lr = (strong == 'l') and 'l' or 'r'
3424   local outer = strong
3425
3426   local new_dir = false
3427   local first_dir = false
3428
3429   local last_lr
3430
3431   local type_n = ''
3432
3433   for item in node.traverse(head) do
3434
3435     -- three cases: glyph, dir, otherwise
3436     if item.id == node.id'glyph' then
3437
3438       local chardata = characters[item.char]
3439       dir = chardata and chardata.d or nil
3440       if not dir then
3441         for nn, et in ipairs(ranges) do
3442           if item.char < et[1] then
3443             break
3444           elseif item.char <= et[2] then
3445             dir = et[3]
3446             break
3447           end
3448         end
3449       end
3450       dir = dir or 'l'

```

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.

```

3451   if new_dir then
3452     attr_dir = 0
3453     for at in node.traverse(item.attr) do
3454       if at.number == luatexbase.registernumber'bbl@attr@dir' then
3455         attr_dir = at.value
3456       end
3457     end
3458     if attr_dir == 1 then
3459       strong = 'r'
3460     elseif attr_dir == 2 then
3461       strong = 'al'
3462     else
3463       strong = 'l'
3464     end

```

```

3465     strong_lr = (strong == 'l') and 'l' or 'r'
3466     outer = strong_lr
3467     new_dir = false
3468 end
3469
3470     if dir == 'nsm' then dir = strong end           -- W1

```

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

```

3471     dir_real = dir           -- We need dir_real to set strong below
3472     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:

```

3473     if strong == 'al' then
3474         if dir == 'en' then dir = 'an' end           -- W2
3475         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
3476         strong_lr = 'r'                               -- W3
3477     end

```

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

```

3478     elseif item.id == node.id'dir' then
3479         new_dir = true
3480         dir = nil
3481     else
3482         dir = nil           -- Not a char
3483     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 behaviour could be changed with a switch in the future. Note in the first branch only `<an>` is relevant if `<al>`.

```

3484     if dir == 'en' or dir == 'an' or dir == 'et' then
3485         if dir ~= 'et' then
3486             type_n = dir
3487         end
3488         first_n = first_n or item
3489         last_n = last_es or item
3490         last_es = nil
3491     elseif dir == 'es' and last_n then -- W3+W6
3492         last_es = item
3493     elseif dir == 'cs' then           -- it's right - do nothing
3494     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
3495         if strong_lr == 'r' and type_n ~= '' then
3496             dir_mark(head, first_n, last_n, 'r')
3497         elseif strong_lr == 'l' and first_d and type_n == 'an' then
3498             dir_mark(head, first_n, last_n, 'r')
3499             dir_mark(head, first_d, last_d, outer)
3500             first_d, last_d = nil, nil
3501         elseif strong_lr == 'l' and type_n ~= '' then
3502             last_d = last_n
3503         end
3504         type_n = ''
3505         first_n, last_n = nil, nil
3506     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:

```

3507   if dir == 'l' or dir == 'r' then
3508     if dir ~= outer then
3509       first_d = first_d or item
3510       last_d = item
3511     elseif first_d and dir ~= strong_lr then
3512       dir_mark(head, first_d, last_d, outer)
3513       first_d, last_d = nil, nil
3514     end
3515   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.

```

3516   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
3517     item.char = characters[item.char] and
3518               characters[item.char].m or item.char
3519   elseif (dir or new_dir) and last_lr ~= item then
3520     local mir = outer .. strong_lr .. (dir or outer)
3521     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
3522       for ch in node.traverse(node.next(last_lr)) do
3523         if ch == item then break end
3524         if ch.id == node.id'glyph' then
3525           ch.char = characters[ch.char].m or ch.char
3526         end
3527       end
3528     end
3529   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).

```

3530   if dir == 'l' or dir == 'r' then
3531     last_lr = item
3532     strong = dir_real          -- Don't search back - best save now
3533     strong_lr = (strong == 'l') and 'l' or 'r'
3534   elseif new_dir then
3535     last_lr = nil
3536   end
3537 end

```

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

```

3538   if last_lr and outer == 'r' then
3539     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
3540       ch.char = characters[ch.char].m or ch.char
3541     end
3542   end
3543   if first_n then
3544     dir_mark(head, first_n, last_n, outer)
3545   end
3546   if first_d then
3547     dir_mark(head, first_d, last_d, outer)
3548   end

```

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

```
3549 return node.prev(head) or head
3550 end
3551 </basic-r>
```

## 14 The ‘nil’ language

This ‘language’ does nothing, except setting the hyphenation patterns to `nohyphenation`. For this language currently no special definitions are needed or available. The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
3552 <*nil>
3553 \ProvidesLanguage{nil}[\langle date \rangle \langle version \rangle Nil language]
3554 \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.

```
3555 \ifx\l@nohyphenation\@undefined
3556   \@nopatterns{nil}
3557   \adddialect\l@nil0
3558 \else
3559   \let\l@nil\l@nohyphenation
3560 \fi
```

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

```
3561 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
\datenil 3562 \let\captionnil\@empty
3563 \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.

```
3564 \ldf@finish{nil}
3565 </nil>
```

## 15 Support for Plain T<sub>E</sub>X (plain.def)

### 15.1 Not renaming `hyphen.tex`

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based T<sub>E</sub>X-format. When asked he responded:

That file name is “sacred”, and if anybody changes it they will cause severe upward/downward compatibility headaches.

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

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to achieve the desired effect, based on the `babel` package. If you load each of them with `iniTeX`, you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`. As these files are going to be read as the first thing `iniTeX` sees, we need to set some category codes just to be able to change the definition of `\input`

```
3566 (*bplain | blplain)
3567 \catcode`\{=1 % left brace is begin-group character
3568 \catcode`\}=2 % right brace is end-group character
3569 \catcode`\#=6 % hash mark is macro parameter character
```

Now let's see if a file called `hyphen.cfg` can be found somewhere on `TeX`'s input path by trying to open it for reading...

```
3570 \openin 0 hyphen.cfg
```

If the file wasn't found the following test turns out true.

```
3571 \ifeof0
3572 \else
```

When `hyphen.cfg` could be opened we make sure that *it* will be read instead of the file `hyphen.tex` which should (according to Don Knuth's ruling) contain the american English hyphenation patterns and nothing else.

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).

```
3573 \let\input
```

Then `\input` is defined to forget about its argument and load `hyphen.cfg` instead.

```
3574 \def\input #1 {%
3575   \let\input\input
3576   \input #1 hyphen.cfg
```

Once that's done the original meaning of `\input` can be restored and the definition of `\input` can be forgotten.

```
3577   \let\input\input
3578 }
3579 \fi
3580 (/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`.

```
3581 (bplain)\input plain.tex
3582 (blplain)\input 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.

```
3583 (bplain)\def\fmtname{babel-plain}
3584 (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.

## 15.2 Emulating some $\LaTeX$ features

The following code duplicates or emulates parts of  $\LaTeX 2_{\epsilon}$  that are needed for `babel`.

```
3585 (*plain)
3586 \def\@empty{}
3587 \def\loadlocalcfg#1{%
```

```

3588 \openin0#1.cfg
3589 \ifeof0
3590 \closein0
3591 \else
3592 \closein0
3593 {\immediate\write16{*****}%
3594 \immediate\write16{* Local config file #1.cfg used}%
3595 \immediate\write16{*}%
3596 }
3597 \input #1.cfg\relax
3598 \fi
3599 \@endoflfd}

```

### 15.3 General tools

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

```

3600 \long\def\@firstofone#1{#1}
3601 \long\def\@firstoftwo#1#2{#1}
3602 \long\def\@secondoftwo#1#2{#2}
3603 \def\@nnil{\@nil}
3604 \def\@gobbletwo#1#2{}
3605 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
3606 \def\@star@or@long#1{%
3607 \@ifstar
3608 {\let\l@ngrel@x\relax#1}%
3609 {\let\l@ngrel@x\long#1}}
3610 \let\l@ngrel@x\relax
3611 \def\@car#1#2\@nil{#1}
3612 \def\@cdr#1#2\@nil{#2}
3613 \let\@typeset@protect\relax
3614 \let\protected@edef\edef
3615 \long\def\@gobble#1{}
3616 \edef\@backslashchar{\expandafter\@gobble\string\}
3617 \def\strip@prefix#1>{}
3618 \def\g@addto@macro#1#2{#2}%
3619 \toks@\expandafter{#1#2}%
3620 \xdef#1{\the\toks@}
3621 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
3622 \def\@nameuse#1{\csname #1\endcsname}
3623 \def\@ifundefined#1{%
3624 \expandafter\ifx\csname#1\endcsname\relax
3625 \expandafter\@firstoftwo
3626 \else
3627 \expandafter\@secondoftwo
3628 \fi}
3629 \def\@expandtwoargs#1#2#3{%
3630 \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
3631 \def\zap@space#1 #2{%
3632 #1%
3633 \ifx#2\@empty\else\expandafter\zap@space\fi
3634 #2}

```

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

```

3635 \ifx\@preamblecmds\undefined
3636 \def\@preamblecmds{}
3637 \fi
3638 \def\@onlypreamble#1{%

```

```

3639 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
3640   \@preamblecmds\do#1}}
3641 \@onlypreamble\@onlypreamble

```

Mimick L<sup>A</sup>T<sub>E</sub>X's \AtBeginDocument; for this to work the user needs to add \begindocument to his file.

```

3642 \def\begindocument{%
3643   \@begindocumenthook
3644   \global\let\@begindocumenthook\@undefined
3645   \def\do##1{\global\let##1\@undefined}%
3646   \@preamblecmds
3647   \global\let\do\noexpand}

3648 \ifx\@begindocumenthook\@undefined
3649   \def\@begindocumenthook{}
3650 \fi
3651 \@onlypreamble\@begindocumenthook
3652 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

We also have to mimick L<sup>A</sup>T<sub>E</sub>X's \AtEndOfPackage. Our replacement macro is much simpler; it stores its argument in \@endofldf.

```

3653 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
3654 \@onlypreamble\AtEndOfPackage
3655 \def\@endofldf{}
3656 \@onlypreamble\@endofldf
3657 \let\bbl@afterlang\@empty
3658 \chardef\bbl@opt@hyphenmap\z@

```

L<sup>A</sup>T<sub>E</sub>X needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default.

```

3659 \ifx\if@files\@undefined
3660   \expandafter\let\csname if@files\expandafter\endcsname
3661     \csname iffalse\endcsname
3662 \fi

```

Mimick L<sup>A</sup>T<sub>E</sub>X's commands to define control sequences.

```

3663 \def\newcommand{\@star@or@long\new@command}
3664 \def\new@command#1{%
3665   \@testopt{\@newcommand#1}0}
3666 \def\@newcommand#1[#2]{%
3667   \@ifnextchar [{\@xargdef#1[#2]}%
3668     {\@argdef#1[#2]}}
3669 \long\def\@argdef#1[#2]#3{%
3670   \@yargdef#1\@ne{#2}{#3}}
3671 \long\def\@xargdef#1[#2][#3]#4{%
3672   \expandafter\def\expandafter#1\expandafter{%
3673     \expandafter\@protected@testopt\expandafter #1%
3674     \csname\string#1\expandafter\endcsname{#3}}%
3675   \expandafter\@yargdef \csname\string#1\endcsname
3676     \tw@{#2}{#4}}
3677 \long\def\@yargdef#1#2#3{%
3678   \@tempcnta#3\relax
3679   \advance \@tempcnta \@ne
3680   \let\@hash@\relax
3681   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
3682   \@tempcntb #2%
3683   \@whilenum\@tempcntb <\@tempcnta
3684     \do{%
3685       \edef\reserved@a{\reserved@a\@hash@the\@tempcntb}%

```

```

3686 \advance\@tempcntb \@ne}%
3687 \let\@hash@##%
3688 \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
3689 \def\providecommand{\@star@or@long\provide@command}
3690 \def\provide@command#1{%
3691 \begingroup
3692 \escapechar@m@ne\xdef\@gtempa{\string#1}%
3693 \endgroup
3694 \expandafter\@ifundefined\@gtempa
3695 {\def\reserved@a{\new@command#1}}%
3696 {\let\reserved@a\relax
3697 \def\reserved@a{\new@command\reserved@a}}%
3698 \reserved@a}%

3699 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
3700 \def\declare@robustcommand#1{%
3701 \edef\reserved@a{\string#1}%
3702 \def\reserved@b{#1}%
3703 \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
3704 \edef#1{%
3705 \ifx\reserved@a\reserved@b
3706 \noexpand\x@protect
3707 \noexpand#1%
3708 \fi
3709 \noexpand\protect
3710 \expandafter\noexpand\csname\bbl@stripslash#1 \endcsname
3711 }%
3712 \expandafter\new@command\csname\bbl@stripslash#1 \endcsname
3713 }
3714 \def\x@protect#1{%
3715 \ifx\protect\@typeset@protect\else
3716 \@x@protect#1%
3717 \fi
3718 }
3719 \def\@x@protect#1\fi#2#3{%
3720 \fi\protect#1%
3721 }

```

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`.

```

3722 \def\bbl@tempa{\csname newif\endcsname\ifin@}
3723 \ifx\in@\undefined
3724 \def\in@#1#2{%
3725 \def\in@##1#1##2##3\in@{%
3726 \ifx\in@##2\in@false\else\in@true\fi}%
3727 \in@#2#1\in@\in@}
3728 \else
3729 \let\bbl@tempa\empty
3730 \fi
3731 \bbl@tempa

```

$\LaTeX$  has a macro to check whether a certain package was loaded with specific options. The command has two extra arguments which are code to be executed in either the true or false case. This is used to detect whether the document needs one of the accents to be activated (activegrave and activeacute). For plain  $\TeX$  we assume that the user wants them to be active by default. Therefore the only thing we do is execute the third argument (the code for the true case).

```
3732 \def\ifpackagewith#1#2#3#4{#3}
```

The L<sup>A</sup>T<sub>E</sub>X macro `\ifl@aded` checks whether a file was loaded. This functionality is not needed for plain T<sub>E</sub>X but we need the macro to be defined as a no-op.

```
3733 \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 L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> versions; just enough to make things work in plain T<sub>E</sub>X environments.

```
3734 \ifx\@tempcnta\@undefined
3735   \csname newcount\endcsname\@tempcnta\relax
3736 \fi
3737 \ifx\@tempcntb\@undefined
3738   \csname newcount\endcsname\@tempcntb\relax
3739 \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`).

```
3740 \ifx\bye\@undefined
3741   \advance\count10 by -2\relax
3742 \fi
3743 \ifx\@ifnextchar\@undefined
3744   \def\@ifnextchar#1#2#3{%
3745     \let\reserved@d=#1%
3746     \def\reserved@a{#2}\def\reserved@b{#3}%
3747     \futurelet\@let@token\@ifnch}
3748   \def\@ifnch{%
3749     \ifx\@let@token\@sptoken
3750       \let\reserved@c\@xifnch
3751     \else
3752       \ifx\@let@token\reserved@d
3753         \let\reserved@c\reserved@a
3754       \else
3755         \let\reserved@c\reserved@b
3756       \fi
3757     \fi
3758     \reserved@c}
3759   \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
3760   \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
3761 \fi
3762 \def\@testopt#1#2{%
3763   \@ifnextchar[#{#1}{#1[#2]}}
3764 \def\@protected@testopt#1{%
3765   \ifx\protect\@typeset@protect
3766     \expandafter\@testopt
3767   \else
3768     \@x@protect#1%
3769   \fi}
3770 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
3771   #2\relax}\fi}
3772 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
3773   \else\expandafter\@gobble\fi{#1}}
```

## 15.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain T<sub>E</sub>X environment.

```
3774 \def\DeclareTextCommand{%
3775   \@dec@text@cmd\providecommand
```

```

3776 }
3777 \def\ProvideTextCommand{%
3778   \@dec@text@cmd\providecommand
3779 }
3780 \def\DeclareTextSymbol#1#2#3{%
3781   \@dec@text@cmd\chardef#1{#2}#3\relax
3782 }
3783 \def\@dec@text@cmd#1#2#3{%
3784   \expandafter\def\expandafter#2%
3785     \expandafter{%
3786       \csname#3-cmd\expandafter\endcsname
3787       \expandafter#2%
3788       \csname#3\string#2\endcsname
3789     }%
3790 %   \let\@ifdefinable\@rc@ifdefinable
3791 \expandafter#1\csname#3\string#2\endcsname
3792 }
3793 \def\@current@cmd#1{%
3794   \ifx\protect\@typeset@protect\else
3795     \noexpand#1\expandafter\@gobble
3796   \fi
3797 }
3798 \def\@changed@cmd#1#2{%
3799   \ifx\protect\@typeset@protect
3800     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
3801       \expandafter\ifx\csname ?\string#1\endcsname\relax
3802         \expandafter\def\csname ?\string#1\endcsname{%
3803           \@changed@x@err{#1}%
3804         }%
3805       \fi
3806     \global\expandafter\let
3807       \csname\cf@encoding \string#1\expandafter\endcsname
3808       \csname ?\string#1\endcsname
3809     \fi
3810     \csname\cf@encoding\string#1%
3811       \expandafter\endcsname
3812   \else
3813     \noexpand#1%
3814   \fi
3815 }
3816 \def\@changed@x@err#1{%
3817   \errhelp{Your command will be ignored, type <return> to proceed}%
3818   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
3819 \def\DeclareTextCommandDefault#1{%
3820   \DeclareTextCommand#1?%
3821 }
3822 \def\ProvideTextCommandDefault#1{%
3823   \ProvideTextCommand#1?%
3824 }
3825 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
3826 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
3827 \def\DeclareTextAccent#1#2#3{%
3828   \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
3829 }
3830 \def\DeclareTextCompositeCommand#1#2#3#4{%
3831   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
3832   \edef\reserved@b{\string##1}%
3833   \edef\reserved@c{%
3834     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%

```

```

3835 \ifx\reserved@b\reserved@c
3836   \expandafter\expandafter\expandafter\ifx
3837     \expandafter\@car\reserved@a\relax\relax\@nil
3838     \@text@composite
3839   \else
3840     \edef\reserved@b##1{%
3841       \def\expandafter\noexpand
3842         \csname#2\string#1\endcsname###1{%
3843           \noexpand\@text@composite
3844             \expandafter\noexpand\csname#2\string#1\endcsname
3845             ###1\noexpand\@empty\noexpand\@text@composite
3846             {##1}%
3847         }%
3848     }%
3849     \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
3850   \fi
3851   \expandafter\def\csname\expandafter\string\csname
3852     #2\endcsname\string#1-\string#3\endcsname{#4}
3853 \else
3854   \errhelp{Your command will be ignored, type <return> to proceed}%
3855   \errmessage{\string\DeclareTextCompositeCommand\space used on
3856     inappropriate command \protect#1}
3857 \fi
3858 }
3859 \def\@text@composite#1#2#3\@text@composite{%
3860   \expandafter\@text@composite@x
3861     \csname\string#1-\string#2\endcsname
3862 }
3863 \def\@text@composite@x#1#2{%
3864   \ifx#1\relax
3865     #2%
3866   \else
3867     #1%
3868   \fi
3869 }
3870 %
3871 \def\@strip@args#1:#2-#3\@strip@args{#2}
3872 \def\DeclareTextComposite#1#2#3#4{%
3873   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
3874   \bgroup
3875     \lccode\@=#4%
3876     \lowercase{%
3877   \egroup
3878     \reserved@a @%
3879   }%
3880 }
3881 %
3882 \def\UseTextSymbol#1#2{%
3883 %   \let\@curr@enc\cf@encoding
3884 %   \@use@text@encoding{#1}%
3885 %   #2%
3886 %   \@use@text@encoding\@curr@enc
3887 }
3888 \def\UseTextAccent#1#2#3{%
3889 %   \let\@curr@enc\cf@encoding
3890 %   \@use@text@encoding{#1}%
3891 %   #2{\@use@text@encoding\@curr@enc\selectfont#3}%
3892 %   \@use@text@encoding\@curr@enc
3893 }

```

```

3894 \def\@use@text@encoding#1{%
3895 %   \edef\font@encoding{#1}%
3896 %   \xdef\font@name{%
3897 %     \csname\curr@fontshape/\font@size\endcsname
3898 %   }%
3899 %   \pickup@font
3900 %   \font@name
3901 %   \@@enc@update
3902 }
3903 \def\DeclareTextSymbolDefault#1#2{%
3904   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
3905 }
3906 \def\DeclareTextAccentDefault#1#2{%
3907   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
3908 }
3909 \def\cf@encoding{OT1}

```

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

```

3910 \DeclareTextAccent{"}{OT1}{127}
3911 \DeclareTextAccent{'}{OT1}{19}
3912 \DeclareTextAccent{^}{OT1}{94}
3913 \DeclareTextAccent{\`}{OT1}{18}
3914 \DeclareTextAccent{\~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for plain  $\TeX$ .

```

3915 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
3916 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
3917 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
3918 \DeclareTextSymbol{\textquoteright}{OT1}{``'}
3919 \DeclareTextSymbol{\i}{OT1}{16}
3920 \DeclareTextSymbol{\ss}{OT1}{25}

```

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

```

3921 \ifx\scriptsize\@undefined
3922   \let\scriptsize\sevenrm
3923 \fi

```

## 15.5 Babel options

The file `babel.def` expects some definitions made in the  $\LaTeX$  style file. So we must provide them at least some predefined values as well some tools to set them (even if not all options are available). There in 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).

```

3924 \let\bbl@opt@shorthands\@nnil
3925 \def\bbl@ifshorthand#1#2#3{#2}%
3926 \ifx\babeloptionstrings\@undefined
3927   \let\bbl@opt@strings\@nnil
3928 \else
3929   \let\bbl@opt@strings\babeloptionstrings
3930 \fi
3931 \def\bbl@tempa{normal}
3932 \ifx\babeloptionmath\bbl@tempa

```

```

3933 \def\bbl@mathnormal{\noexpand\textormath}
3934 \fi
3935 \def\BabelStringsDefault{generic}
3936 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
3937 \let\bbl@afterlang\relax
3938 \let\bbl@language@opts\@empty
3939 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
3940 \def\AfterBabelLanguage#1#2{}
3941 \
```

## 16 Acknowledgements

I would like to thank all who volunteered as  $\beta$ -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

## References

- [1] Huda Smitsluijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Donald E. Knuth, *The T<sub>E</sub>Xbook*, Addison-Wesley, 1986.
- [3] Leslie Lamport, *L<sup>A</sup>T<sub>E</sub>X, A document preparation System*, Addison-Wesley, 1986.
- [4] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*. SDU Uitgeverij ('s-Gravenhage, 1988).
- [5] Hubert Partl, *German T<sub>E</sub>X*, *TUGboat* 9 (1988) #1, p. 70–72.
- [6] Leslie Lamport, in: T<sub>E</sub>Xhax Digest, Volume 89, #13, 17 February 1989.
- [7] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national L<sup>A</sup>T<sub>E</sub>X styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [8] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [9] Joachim Schrod, *International L<sup>A</sup>T<sub>E</sub>X is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [10] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using L<sup>A</sup>T<sub>E</sub>X*, Springer, 2002, p. 301–373.