

# Babel

Version 3.22.1381

2018/08/27

*Original author*

Johannes L. Braams

*Current maintainer*

Javier Bezos

The standard distribution of  $\text{\LaTeX}$  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  $\text{\LaTeX}$  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  $\text{\TeX}$  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 $\text{\TeX}$  and Lua $\text{\TeX}$ ) 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 . . . . .	10
1.10	Shorthands . . . . .	11
1.11	Package options . . . . .	14
1.12	The base option . . . . .	16
1.13	ini files . . . . .	17
1.14	Selecting fonts . . . . .	23
1.15	Modifying a language . . . . .	24
1.16	Creating a language . . . . .	25
1.17	Digits . . . . .	27
1.18	Getting the current language name . . . . .	27
1.19	Hyphenation tools . . . . .	27
1.20	Selecting scripts . . . . .	29
1.21	Selecting directions . . . . .	29
1.22	Language attributes . . . . .	33
1.23	Hooks . . . . .	33
1.24	Languages supported by babel with ldf files . . . . .	35
1.25	Tips, workarounds, know issues and notes . . . . .	36
1.26	Current and future work . . . . .	37
1.27	Tentative and experimental code . . . . .	38
<b>2</b>	<b>Loading languages with language.dat</b>	<b>39</b>
2.1	Format . . . . .	39
<b>3</b>	<b>The interface between the core of babel and the language definition files</b>	<b>40</b>
3.1	Guidelines for contributed languages . . . . .	41
3.2	Basic macros . . . . .	42
3.3	Skeleton . . . . .	43
3.4	Support for active characters . . . . .	44
3.5	Support for saving macro definitions . . . . .	44
3.6	Support for extending macros . . . . .	44
3.7	Macros common to a number of languages . . . . .	44
3.8	Encoding-dependent strings . . . . .	45
<b>4</b>	<b>Changes</b>	<b>48</b>
4.1	Changes in babel version 3.9 . . . . .	48
4.2	Changes in babel version 3.7 . . . . .	49
<b>II</b>	<b>The code</b>	<b>50</b>
<b>5</b>	<b>Identification and loading of required files</b>	<b>50</b>

6	locale <b>directory</b>	<b>50</b>
7	<b>Tools</b>	<b>51</b>
7.1	Multiple languages . . . . .	54
8	<b>The Package File (<math>\LaTeX</math>, babel.sty)</b>	<b>55</b>
8.1	base . . . . .	56
8.2	key=value options and other general option . . . . .	57
8.3	Conditional loading of shorthands . . . . .	58
8.4	Language options . . . . .	60
9	<b>The kernel of Babel (babel.def, common)</b>	<b>62</b>
9.1	Tools . . . . .	63
9.2	Hooks . . . . .	65
9.3	Setting up language files . . . . .	67
9.4	Shorthands . . . . .	69
9.5	Language attributes . . . . .	78
9.6	Support for saving macro definitions . . . . .	80
9.7	Short tags . . . . .	81
9.8	Hyphens . . . . .	81
9.9	Multiencoding strings . . . . .	83
9.10	Macros common to a number of languages . . . . .	89
9.11	Making glyphs available . . . . .	89
9.11.1	Quotation marks . . . . .	89
9.11.2	Letters . . . . .	90
9.11.3	Shorthands for quotation marks . . . . .	91
9.11.4	Umlauts and tremas . . . . .	92
9.12	Layout . . . . .	93
9.13	Creating languages . . . . .	94
10	<b>The kernel of Babel (babel.def, only <math>\LaTeX</math>)</b>	<b>102</b>
10.1	The redefinition of the style commands . . . . .	102
10.2	Cross referencing macros . . . . .	102
10.3	Marks . . . . .	105
10.4	Preventing clashes with other packages . . . . .	106
10.4.1	ifthen . . . . .	106
10.4.2	varioref . . . . .	107
10.4.3	hhline . . . . .	108
10.4.4	hyperref . . . . .	108
10.4.5	fancyhdr . . . . .	108
10.5	Encoding and fonts . . . . .	109
10.6	Basic bidi support . . . . .	111
10.7	Local Language Configuration . . . . .	113
11	<b>Multiple languages (switch.def)</b>	<b>114</b>
11.1	Selecting the language . . . . .	115
11.2	Errors . . . . .	123
12	<b>Loading hyphenation patterns</b>	<b>124</b>
13	<b>Font handling with fontspec</b>	<b>129</b>

<b>14</b>	<b>Hooks for XeTeX and LuaTeX</b>	<b>132</b>
14.1	XeTeX . . . . .	132
14.2	Layout . . . . .	134
14.3	LuaTeX . . . . .	137
14.4	Layout . . . . .	142
14.5	Auto bidi with basic-r . . . . .	144
<b>15</b>	<b>The ‘nil’ language</b>	<b>155</b>
<b>16</b>	<b>Support for Plain T<sub>E</sub>X (plain.def)</b>	<b>155</b>
16.1	Not renaming hyphen.tex . . . . .	155
16.2	Emulating some L <sup>A</sup> T <sub>E</sub> X features . . . . .	156
16.3	General tools . . . . .	157
16.4	Encoding related macros . . . . .	160
<b>17</b>	<b>Acknowledgements</b>	<b>163</b>

## Troubleshooting

Paragraph ended before \UTFviii@three@octets was complete . . . . .	4
No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the format . . . . .	5
You are loading directly a language style . . . . .	7
Unknown language ‘LANG’ . . . . .	7
Argument of \language@active@arg” has an extra } . . . . .	11

# 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 on <https://github.com/latex3/latex2e/issues>, and note just complaining on an e-mail list or a web forum is usually of little help.
- 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.
- The first sections describe the traditional way of loading a language (with ldf files). The alternative way based on ini files, which complements the previous one (it will *not* replace it), is described below.

## 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 (however, the package inputenc may be omitted with  $\LaTeX \geq 2018-04-01$  if the encoding is UTF-8):

```
\documentclass{article}

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

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

**TROUBLESHOOTING** A common source of trouble is a wrong setting of the input encoding. Very often you will get the following somewhat cryptic error:

```
! Paragraph ended before \UTFviii@three@octets was complete.
```

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.

**TROUBLESHOOTING** The following warning is about hyphenation patterns, which are not under the direct control of `babel`:

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

The document will be typeset, but very likely the text will not be correctly hyphenated. Some languages may be raising this warning wrongly (because they are not hyphenated); it is a bug to be fixed – just ignore it. See the manual of your distribution (MacTeX, MikTeX, T<sub>E</sub>XLive, etc.) for further info about how to configure it.

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

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

would tell L<sup>A</sup>T<sub>E</sub>X that the document would be written in two languages, Dutch and English, and that English would be the first language in use, and the main one.

You can also set the main language explicitly:

```
\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 `\language` (in particular, shorthands, captions and date are not activated). If you need to define boxes and the like in the preamble, you might want to use some of the language selectors described below.

To switch the language there are two basic macros, described below in detail: `\selectlanguage` is used for blocks of text, while `\foreignlanguage` is for chunks of text inside paragraphs.

**EXAMPLE** A full bilingual document follows. The main language is french, which is activated when the document begins. The package `inputenc` may be omitted with  $\LaTeX \geq 2018-04-01$  if the encoding is UTF-8.

```
\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 behavior of some languages can be modified when loading `babel` by means of *modifiers*. They are set after the language name, and are prefixed with a dot (only when the language is set as package option – neither global options nor the `main` key 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 lualatex

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

The Latin script is covered by default in current  $\LaTeX$  (provided the document encoding is UTF-8), because the font loader is preloaded and the font is switched to `lmroman`. Other scripts require loading `fontspec`. You may want to set the font attributes with `fontspec`, too.

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

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

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

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

---

<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”.



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

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

## 1.6 Plain

In 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:

---

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

```
\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).

## 1.8 Auxiliary language selectors

**\begin{otherlanguage}**     $\langle language \rangle \dots \end{otherlanguage}$

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\*}**     $\langle language \rangle \dots \end{otherlanguage*}$

Same as `\foreignlanguage` but as environment. Spaces after the environment are *not* ignored.

This environment was originally intended for intermixing left-to-right typesetting with right-to-left typesetting in engines not supporting a change in the writing direction inside a line. However, by default it never complied with the documented behavior and it is just a version as environment of `\foreignlanguage`, except when the option `bidi` is set – in this case, `\foreignlanguage` emits a `\leavevmode`, while `otherlanguage*` does not.

**\begin{hyphenrules}**     $\langle language \rangle \dots \end{hyphenrules}$

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**  $\{\langle tag1 \rangle = \langle language1 \rangle, \langle tag2 \rangle = \langle language2 \rangle, \dots\}$

**New 3.9i** In multilingual documents with many language switches the commands above can be cumbersome. With this tool shorter names can be defined. It adds nothing really new – it is just syntactical sugar.

It defines  $\text{\text{<tag1>\{<text>\}}$  to be  $\text{\foreignlanguage{<language1>\{<text>\}}$ , and  $\text{\begin{<tag1>\{<text>\}}$  to be  $\text{\begin{other language*}{<language1>\{<text>\}}$ , and so on. Note  $\text{\<tag1>}$  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  $\text{\babeltags{finnish = finnish}}$  is legitimate – it defines  $\text{\text{finnish}}$  and  $\text{\begin{finnish}}$  (and, of course,  $\text{\begin{finnish}}$ ).

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

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

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

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

Of course,  $\text{\TeX}$  can do it for you. To avoid switching the language all the while,  $\text{\babelensure}$  redefines the captions for a given language to wrap them with a selector:

```
\babelensure{polish}
```

By default only the basic captions and  $\text{\today}$  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:

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

```
\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; (2) in some languages shorthands such as ! are used to insert the right amount of white space; (3) several kinds of discretionary and breaks can be inserted easily with "-", "=", etc. The package `inputenc` as well as `xetex` and `luatex` have alleviated entering non-ASCII characters, but minority languages and some kinds of text can still require characters not directly available on the keyboards (and sometimes not even as separated or precomposed Unicode characters). As to the point 2, now `pdfTeX` provides `\knbcode`, and `luatex` can manipulate the glyph list. Tools for point 3 can be still very useful in general. There are 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.

**\useshorthands** `*{⟨char⟩}`

The command `\useshorthands` 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 `\useshorthands*{⟨char⟩}` is provided, which makes sure shorthands are always activated.

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

**\defineshorthand** `[⟨language⟩,⟨language⟩,...]{⟨shorthand⟩}{⟨code⟩}`

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

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

User shorthands override language ones, which in turn override system shorthands. Language-dependent user shorthands (new in 3.9) take precedence over “normal” user shorthands.

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

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

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

```
\defineshorthand[*polish,*portuguese]{"-}{\babelhyphen{repeat}}
```

Here, options with `*` set a language-dependent user shorthand, which means the generic one above only applies for the rest of languages; without `*` they would (re)define the language shorthands instead, which are overridden by user ones.

Now, you have a single unified shorthand (`"-`), with a content-based meaning (‘compound word hyphen’) whose visual behavior is that expected in each context.

**\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 easily type phonetic characters with `tipa`:

```
\newcommand{\myipa}[1]{\{\languageshorthands{none}\tipaencoding#1}}
```

**\babelshorthand**  $\{\langle shorthand \rangle\}$

With this command you can use a shorthand even if (1) not activated in shorthands (in this case only shorthands for the current language are taken into account, ie, not user shorthands), (2) turned off with `\shorthandoff` or (3) deactivated with the internal `\bbl@deactivate`; for example, `\babelshorthand{"u}` or `\babelshorthand{:}`. (You can conveniently define your own macros, or even your own user shorthands provided they do not overlap.)

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

<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

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

**Basque** " ' ~

**Breton** : ; ? !

**Catalan** " ' `

**Czech** " -

**Esperanto** ^

**Estonian** " ~

**French** (all varieties) : ; ? !

**Galician** " . ' ~ < >

**Greek** ~

**Hungarian** `

**Kurmanji** ^

**Latin** " ^ =

**Slovak** " ^ ' -

**Spanish** " . < > '

**Turkish** : ! =

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

<b>KeepShorthandsActive</b>	Tells babel not to deactivate shorthands after loading a language file, so that they are also available in the preamble.
<b>activeacute</b>	For some languages babel supports this options to set ' as a shorthand in case it is not done by default.
<b>activegrave</b>	Same for `.
<b>shorthands=</b>	$\langle char \rangle \langle char \rangle \dots$   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  $\TeX$  before they are passed to the package and therefore they will not be recognized); however, t is provided for the common case of ~ (as well as c for not so common case of the comma). With shorthands=off no language shorthands are defined, As some languages use this mechanism for tools not available otherwise, a macro \babelshorthand is defined, which allows using them; see above.

**safe=** none | ref | bib

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

Some  $\LaTeX$  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).

- 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 spoilt by an unexpected `.cfg` file. However, if the key `config` is set, this file is loaded.
- showlanguages** Prints to the log the list of languages loaded when the format was created: number (remember dialects can share it), name, hyphenation file and exceptions file.
- nocase** New 3.9! Language settings for uppercase and lowercase mapping (as set by `\SetCase`) are ignored. Use only if there are incompatibilities with other packages.
- silent** New 3.9! 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 `\MakeUpper case` and the like (this feature misuses some internal  $\LaTeX$  tools, so use it only as a last resort).
- hyphenmap=** off | main | select | other | other\*

---

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



**New 3.9g** Sets the behavior 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>  
**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. 1.21.

**layout=**

**New 3.16** Selects which layout elements are adapted in bidi documents. See sec. 1.21.

## 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** `{⟨option-name⟩}{⟨code⟩}`

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

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

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

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

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

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

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

<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 behavior it might be added. On the other hand, `other` is provided even if I [JBL] think it isn't really useful, but who knows.

```
\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 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 (in other words, \babelprovide is mainly intended for auxiliary tasks).

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

```
\documentclass{book}

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

\babelfont{rm}{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>ul</sup>	bem	Bemba
agq	Aghem	bez	Bena
ak	Akan	bg	Bulgarian <sup>ul</sup>
am	Amharic <sup>ul</sup>	bm	Bambara
ar	Arabic <sup>ul</sup>	bn	Bangla <sup>ul</sup>
ar-DZ	Arabic <sup>ul</sup>	bo	Tibetan <sup>u</sup>
ar-MA	Arabic <sup>ul</sup>	brx	Bodo
ar-SY	Arabic <sup>ul</sup>	bs-Cyrl	Bosnian
as	Assamese	bs-Latn	Bosnian <sup>ul</sup>
asa	Asu	bs	Bosnian <sup>ul</sup>
ast	Asturian <sup>ul</sup>	ca	Catalan <sup>ul</sup>
az-Cyrl	Azerbaijani	ce	Chechen
az-Latn	Azerbaijani	cgg	Chiga
az	Azerbaijani <sup>ul</sup>	chr	Cherokee
bas	Basaa	ckb	Central Kurdish
be	Belarusian <sup>ul</sup>	cs	Czech <sup>ul</sup>

cy	Welsh <sup>ul</sup>	hy	Armenian
da	Danish <sup>ul</sup>	ia	Interlingua <sup>ul</sup>
dav	Taita	id	Indonesian <sup>ul</sup>
de-AT	German <sup>ul</sup>	ig	Igbo
de-CH	German <sup>ul</sup>	ii	Sichuan Yi
de	German <sup>ul</sup>	is	Icelandic <sup>ul</sup>
dje	Zarma	it	Italian <sup>ul</sup>
dsb	Lower Sorbian <sup>ul</sup>	ja	Japanese
dua	Duala	jgo	Ngomba
dyo	Jola-Fonyi	jmc	Machame
dz	Dzongkha	ka	Georgian <sup>ul</sup>
ebu	Embu	kab	Kabyle
ee	Ewe	kam	Kamba
el	Greek <sup>ul</sup>	kde	Makonde
en-AU	English <sup>ul</sup>	kea	Kabuverdianu
en-CA	English <sup>ul</sup>	khq	Koyra Chiini
en-GB	English <sup>ul</sup>	ki	Kikuyu
en-NZ	English <sup>ul</sup>	kk	Kazakh
en-US	English <sup>ul</sup>	kkj	Kako
en	English <sup>ul</sup>	kl	Kalaallisut
eo	Esperanto <sup>ul</sup>	kln	Kalenjin
es-MX	Spanish <sup>ul</sup>	km	Khmer
es	Spanish <sup>ul</sup>	kn	Kannada <sup>ul</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>1</sup>	mg	Malagasy
ha	Hausa	mgh	Makhuwa-Meetto
haw	Hawaiian	mgo	Meta'
he	Hebrew <sup>ul</sup>	mk	Macedonian <sup>ul</sup>
hi	Hindi <sup>u</sup>	ml	Malayalam <sup>ul</sup>
hr	Croatian <sup>ul</sup>	mn	Mongolian
hsb	Upper Sorbian <sup>ul</sup>	mr	Marathi <sup>ul</sup>
hu	Hungarian <sup>ul</sup>	ms-BN	Malay <sup>1</sup>

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

---

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

---

aghem	centralatlastamazight
akan	centralkurdish
albanian	chechen
american	cherokee
amharic	chiga
arabic	chinese-hans-hk
arabic-algeria	chinese-hans-mo
arabic-DZ	chinese-hans-sg
arabic-morocco	chinese-hans
arabic-MA	chinese-hant-hk
arabic-syria	chinese-hant-mo
arabic-SY	chinese-hant
armenian	chinese-simplified-hongkongsarchina
assamese	chinese-simplified-macausarchina
asturian	chinese-simplified-singapore
asu	chinese-simplified
australian	chinese-traditional-hongkongsarchina
austrian	chinese-traditional-macausarchina
azerbaijani-cyrillic	chinese-traditional
azerbaijani-cyrl	chinese
azerbaijani-latin	cognian
azerbaijani-latn	cornish
azerbaijani	croatian
bafia	czech
bambara	danish
basaa	duala
basque	dutch
belarusian	dzongkha
bemba	embu
ben	english-au
bengali	english-australia
bodo	english-ca
bosnian-cyrillic	english-canada
bosnian-cyrl	english-gb
bosnian-latin	english-newzealand
bosnian-latn	english-nz
bosnian	english-unitedkingdom
brazilian	english-unitedstates
breton	english-us
british	english
bulgarian	esperanto
burmese	estonian
canadian	ewe
cantonese	ewondo
catalan	faroes

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

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

---

## 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`).

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]{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.

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



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

```
\belfont{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}
\belfont[hindi]{rm}{Shobhika}
```

This makes sure the OpenType script for Devanagari is `deva` and not `dev2` (luatex does not detect automatically the correct script<sup>14</sup>).

**NOTE** Directionality is a property affecting margins, intonation, 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 `\belfont` if omitted, but the opposite is not true. See the note above for the reasons of this behavior.

**WARNING** Do not use `\setxxxxfont` and `\belfont` at the same time. `\belfont` follows the standard  $\text{\LaTeX}$  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 `\belfont` just does *not* work (nor the standard `\xxdefault`, for that matter).

## 1.15 Modifying a language

Modifying the behavior 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\captionsenglish{%
  \renewcommand\contentsname{Foo}%
}
```

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

---

<sup>14</sup>And even with the correct code some fonts could be rendered incorrectly by `fontspec`, so double check the results. `xetex` fares better, but some font are still problematic.

- 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\extrasrussian{\mymacro}
```

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

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

## 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.

**New 3.23** It may be used without a value. In such a case, the ini file set in the corresponding babel-<language>.tex (where <language> is the last argument in \babelprovide) is imported. See the list of recognized languages above.

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=** *<language-list>*

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

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

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

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

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

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

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

**script=** *<script-name>*

**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 Digits

**New 3.20** A few ini files define a field named `digits.native`. When it is present, two macros are created: `\<language>digits` and `\<language>counter` (only xetex and luatex). With the first, a string of ‘Latin’ digits are converted to the native digits of that language; the second takes a counter name as argument. With option `maparabic` in `\babelprovide`, `\arabic` is redefined to produce the native digits (this is done *globally*, to avoid inconsistencies in, for example, page numbering). For example:

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

Languages providing native digits in all or some variants are *ar, as, bn, bo, brx, ckb, dz, fa, gu, hi, km, kn, kok, ks, lo, lrc, ml, mr, my, mzn, ne, or, pa, ps, ta, te, th, ug, ur, uz, vai, yue, zh*.

## 1.18 Getting the current language name

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

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

`\iflanguage`  $\{\langle$ *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 `\language` also applies here – use `iflang` instead of `\iflanguage` if possible.

## 1.19 Hyphenation tools

`\babelhyphen`  $\ast\{\langle$ *type* $\rangle\}$

`\babelhyphen` `*{<text>}`

**New 3.9a** It is customary to classify hyphens in two types: (1) *explicit* or *hard hyphens*, which in  $\TeX$  are entered as `-`, and (2) *optional* or *soft hyphens*, which are entered as `\-`. Strictly, a *soft hyphen* is not a hyphen, but just a breaking opportunity or, in  $\TeX$  terms, a “discretionary”; a *hard hyphen* is a hyphen with a breaking opportunity after it. A further type is a *non-breaking hyphen*, a hyphen without a breaking opportunity.

In  $\TeX$ , `-` and `\-` forbid further breaking opportunities in the word. This is the desired behavior very often, but not always, and therefore many languages provide shorthands for these cases. Unfortunately, this has not been done consistently: for example, `-` in Dutch, Portugese, 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, Portugese 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 `\babelnullhyphen`; (3) a break after the hyphen is forbidden if preceded by a glue  $>0$  pt (at the beginning of a word, provided it is not immediately preceded by, say, a parenthesis).

`\babelhyphenation` [`<language>`], [`<language>`], ... [`<exceptions>`]

**New 3.9a** Sets hyphenation exceptions for the languages given or, without the optional argument, for *all* languages (eg, proper nouns or common loan words, and of course monolingual documents). 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>15</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.

## 1.20 Selecting scripts

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

Some languages sharing the same script define macros to switch it (eg, `\textcyrillic`), but be aware they may also set the language to a certain default. Even the babel core defined `\textlatin`, but it was somewhat buggy because in some cases it messed up encodings and fonts (for example, if the main latin encoding was LY1), and therefore it has been deprecated.<sup>17</sup>

`\ensureascii` { *<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.

## 1.21 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%).

<sup>15</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>16</sup>The so-called Unicode fonts do not improve the situation either. So, a font suited for Vietnamese is not necessarily suited for, say, the romanization of Indic languages, and the fact it contains glyphs for Modern Greek does not mean it includes them for Classic Greek.

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

**WARNING** The current code for **text** in luatex should be considered essentially stable, but, of course, it is not bug free and there could be improvements in the future, because setting bidi text has many subtleties (see for example <https://www.w3.org/TR/html-bidi/>). A basic stable version for other engines must wait very likely until (Northern) Winter. This applies to text, but **graphical** elements, including the picture environment and PDF or PS based graphics, are not yet correctly handled. Also, indexes and the like are under study, as well as math.

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

There are some package options controlling bidi writing.

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

**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. **New 3.19** Finally, `basic` supports both L and R text. (They are named `basic` mainly because they only consider the intrinsic direction of scripts and weak directionality.) There are samples on GitHub, under `/required/babel/samples`. See particularly `lua-bidibasic.tex` and `lua-secenum.tex`.

**EXAMPLE** The following text comes from the Arabic Wikipedia (article about Arabia). Copy-pasting some text from the Wikipedia is a good way to test this feature. Remember `basic-r` is available in luatex only.<sup>18</sup>

```
\documentclass{article}

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

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

<sup>18</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. Have a look at the workaround available on GitHub, under `/required/babel/samples`

```

\documentclass{book}

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

\babelprovide[mapfont=direction]{arabic}

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

\begin{document}

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

\end{document}

```

What `mapfont=direction` means is, ‘when a character has the same direction as the script for the “provided” language (arabic in this case), then change its font to that set for this language’ (here defined via `*arabic`, because Crimson does not provide Arabic letters). Note Hebrew and Arabic have different directions (internally ‘r’ and ‘al’, respectively).

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

```

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

```

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

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

**New 3.16** *To be expanded.* Selects which layout elements are adapted in bidi documents, including some text elements. You may use several options with a comma-separated list (eg, `layout=counters.contents.sectioning`). This list will be expanded in future releases (tables, captions, etc.). Note not all options are required by all engines.

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

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

With counters, `\arabic` is not only considered L text always (with `\babelsublr`, see below), but also an “isolated” block which does not interact with the surrounding chars.



So, while 1.2 in R text is rendered in that order with `bidi=basic` (as a decimal number), in `\arabic{c1}.\arabic{c2}` the visual order is `c2.c1`. Of course, you may always adjust the order by changing the language, if necessary.<sup>19</sup>

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

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

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

**footnotes** not required in monolingual documents, but it may be useful in multilingual documents in all engines; you may use alternatively `\BabelFootnote` described below (what this options does exactly is also explained there).

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

**tabular** required in luatex for R tabular (it has been tested only with simple tables, so expect some readjustments in the future); ignored in pdftex or xetex (which will not support a similar option in the short term) [New 3.18](#) ,

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

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

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

Any `\babelsublr` in *explicit* L mode is ignored. However, with `bidi=basic` and *implicit* L, it first returns to R and then switches to explicit L. To clarify this point, consider, in an R context:

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

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

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

**`\BabelPatchSection`** `{\section-name}`

Mainly for bidi text, but it 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 (it also “isolates” the page number in heads, for a proper bidi behavior), but with this command you can set them individually if necessary (but note then tocs and marks are not touched).

**`\BabelFootnote`** `{\cmd}{\local-language}{\before}{\after}`

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

**New 3.17** Something like:

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

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

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

but the footnote itself is typeset in the main language (to unify its direction). In addition, `\parsfootnotetext` is defined. The option `footnotes` just does the following:

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

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

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

```
\BabelFootnote{\enfootnote}{english}\{({})}
```

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

## 1.22 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).

## 1.23 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{name}`, `\DisableBabelHook{name}`.

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

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

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

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

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

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

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

**stopcommands** Used to reset the 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).

## 1.24 Languages supported by babel with ldf files

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

<b>Afrikaans</b>	afrikaans
<b>Azerbaijani</b>	azerbaijani
<b>Basque</b>	basque
<b>Breton</b>	breton
<b>Bulgarian</b>	bulgarian
<b>Catalan</b>	catalan
<b>Croatian</b>	croatian
<b>Czech</b>	czech
<b>Danish</b>	danish
<b>Dutch</b>	dutch
<b>English</b>	english, USenglish, american, UKenglish, british, canadian, australian, newzealand
<b>Esperanto</b>	esperanto
<b>Estonian</b>	estonian
<b>Finnish</b>	finnish
<b>French</b>	french, francais, canadien, acadian
<b>Galician</b>	galician
<b>German</b>	austrian, german, germanb, ngerman, naustrian
<b>Greek</b>	greek, polutonikogreek
<b>Hebrew</b>	hebrew
<b>Icelandic</b>	icelandic
<b>Indonesian</b>	bahasa, indonesian, indon, bahasai
<b>Interlingua</b>	interlingua
<b>Irish Gaelic</b>	irish
<b>Italian</b>	italian
<b>Latin</b>	latin
<b>Lower Sorbian</b>	lowersorbian
<b>Malay</b>	bahasam, malay, melayu
<b>North Sami</b>	samin
<b>Norwegian</b>	norsk, nynorsk
<b>Polish</b>	polish
<b>Portuguese</b>	portuges, portuguese, brazilian, brazil
<b>Romanian</b>	romanian
<b>Russian</b>	russian
<b>Scottish Gaelic</b>	scottish
<b>Spanish</b>	spanish
<b>Slovakian</b>	slovak
<b>Slovenian</b>	slovene
<b>Swedish</b>	swedish
<b>Serbian</b>	serbian
<b>Turkish</b>	turkish
<b>Ukrainian</b>	ukrainian
<b>Upper Sorbian</b>	upporsorbian
<b>Welsh</b>	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  $\text{\LaTeX}$ .

**NOTE** Please, for info about the support in `luatex` for some complex scripts, see the wiki, on <https://github.com/latex3/latex2e/wiki/Babel:-Remarks-on-the-luatex-support-for-some-scripts>.

## 1.25 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`),  $\text{\LaTeX}$  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\extrasrussian{\inputencoding{koi8-r}}
```

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

- For the hyphenation to work correctly, `lccodes` cannot change, because  $\text{\TeX}$  only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.<sup>20</sup> So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of  $\text{\TeX}$ , 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.

<sup>20</sup>This explains why  $\text{\LaTeX}$  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.

- Babel does not take into account `\normalsfcodes` and (non-)French spacing is not always properly (un)set by languages. However, problems are unlikely to happen and therefore this part remains untouched in version 3.9 (but it is in the ‘to do’ list).
- Using a character mathematically active (ie, with math code "8000) as a shorthand can make  $\TeX$  enter in an infinite loop in some rare cases. (Another issue in the ‘to do’ list, although there is a partial solution.)

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

**csquotes** Logical markup for quotes.

**iflang** Tests correctly the current language.

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

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

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

**biblatex** Programmable bibliographies and citations.

**bicaption** Bilingual captions.

**babelbib** Multilingual bibliographies.

**microtype** Adjusts the typesetting according to some languages (kerning and spacing).  
Ligatures can be disabled.

**substitutefont** Combines fonts in several encodings.

**mkpattern** Generates hyphenation patterns.

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

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

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

## 1.26 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{babel}

\babelprovide[import, 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

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>21</sup>. But that is the easy part, because they don't require modifying the  $\TeX$  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.

## 1.27 Tentative and experimental code

### Old stuff

A couple of tentative macros were provided by babel ( $\geq 3.9g$ ) with a partial solution for “Unicode” fonts. These macros are now deprecated — use `\babelfont`. A short description follows, for reference:

- `\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}

```

<sup>21</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.

**Bidi writing** in `luatex` is under development, but a basic implementation is almost finished. On the other hand, in `xetex` it is taking its first steps. The latter engine poses quite different challenges. An option to manage document layout in `luatex` (lists, footnotes, etc.) is almost finished, but `xetex` required more work.

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 (`pdfTEX`, `xetex`, `ε-TEX`, the main exception being `luatex`) require hyphenation patterns to be preloaded when a format is created (eg, `LATEX`, `XeLATEX`, `pdfLATEX`). `babel` provides a tool which has become standard in many distributions and based on a “configuration file” named `language.dat`. The exact way this file is used depends on the distribution, so please, read the documentation for the latter (note also some distributions generate the file with some tool).

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

### 2.1 Format

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

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

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

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

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

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

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

<sup>23</sup>The loader for `lua(e)tex` is slightly different as it's not based on `babel` but on `etex.src`. Until 3.9p it just didn't work, but thanks to the new code it works by reloading the data in the `babel` way, i.e., with `language.dat`.

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

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



With the previous settings, if the encoding when the language is selected is T1 then the patterns in `hyphenT1.ger` are used, but otherwise use those in `hyphen.ger` (note the encoding 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.

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

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

The following assumptions are made:

- Some of the language-specific definitions might be used by plain  $\text{T}_{\text{E}}\text{X}$  users, so the files have to be coded so that they can be read by both  $\text{\LaTeX}$  and plain  $\text{T}_{\text{E}}\text{X}$ . The current format can be checked by looking at the value of the macro `\fmtname`.
- The common part of the babel system redefines a number of macros and environments (defined previously in the document style) to put in the names of macros that replace the previously hard-wired texts. These macros have to be defined in the language definition files.
- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are `\<lang>hyphenmins`, `\captions{lang}`, `\date{lang}`, `\extras{lang}` and `\noextras{lang}` (the last two may be left empty); where `<lang>` is either the name of the language definition file or the name of the  $\text{\LaTeX}$  option that is to be used. These macros and their functions are discussed below. You must define all or none for a language (or a dialect); defining, say, `\date{lang}` but not `\captions{lang}` does not raise an error but can lead to unexpected results.
- When a language definition file is loaded, it can define `\l@<lang>` to be a dialect of `\language0` when `\l@<lang>` is undefined.
- Language names must be all lowercase. If an unknown language is selected, babel will attempt setting it after lowercasing its name.
- The semantics of modifiers is not defined (on purpose). In most cases, they will just be simple separated options (eg, `spanish`), but a language might require, say, a set of options organized as a tree with suboptions (in such a case, the recommended separator is `/`).

Some recommendations:

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

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

### 3.1 Guidelines for contributed languages

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 your 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.

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

## 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 behavior of the babel system is to define this language as a ‘dialect’ of the language for which the patterns were loaded as `\language0`. Here “language” is used in the  $\TeX$  sense of set of hyphenation patterns.

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

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

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

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

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

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

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

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

**\bbl@declare@tribute** This is a command to be used in the language definition files for declaring a language attribute. It takes three arguments: the name of the language, the attribute to be defined, and the code to be executed when the attribute is to be used.

**\main@language** To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use `\main@language` instead of `\selectlanguage`. This will just store the name of the language, and the proper language will be activated at the start of the document.

**\ProvidesLanguage** The macro `\ProvidesLanguage` should be used to identify the language definition files. Its syntax is similar to the syntax of the  $\TeX$  command `\ProvidesPackage`.

**\LdfInit** The macro `\LdfInit` performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the `@`-sign, preventing the `.ldf` file from being processed twice, etc.

**\ldf@quit** The macro `\ldf@quit` does work needed if a `.ldf` file was processed earlier. This includes resetting the category code of the `@`-sign, preparing the language to be activated at `\begin{document}` time, and ending the input stream.

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

**\loadlocalcfg** After processing a language definition file,  $\TeX$  can be instructed to load a local

configuration file. This file can, for instance, be used to add strings to `\captions<lang>` to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by `\ldf@finish`.

`\substitutefontfamily`

(Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This .fd file will instruct L<sup>A</sup>T<sub>E</sub>X 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]
\LdfInit{<language>}{captions<language>}

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

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

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

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

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

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

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

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

\EndBabelCommands

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

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

### 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.

<code>\initiate@active@char</code>	The internal macro <code>\initiate@active@char</code> is used in language definition files to instruct $\TeX$ to give a character the category code ‘active’. When a character has been made active it will remain that way until the end of the document. Its definition may vary.
<code>\bbl@activate</code> <code>\bbl@deactivate</code>	The command <code>\bbl@activate</code> is used to change the way an active character expands. <code>\bbl@activate</code> ‘switches on’ the active behavior of the character. <code>\bbl@deactivate</code> lets the active character expand to its former (mostly) non-active self.
<code>\declare@shorthand</code>	The macro <code>\declare@shorthand</code> 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. <code>~</code> or <code>"a</code> ; and the code to be executed when the shorthand is encountered. (It does <i>not</i> raise an error if the shorthand character has not been “initiated”.)
<code>\bbl@add@special</code> <code>\bbl@remove@special</code>	The $\TeX$ book states: “Plain $\TeX$ includes a macro called <code>\dospecials</code> that is essentially a set 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 <code>\dospecial</code> . $\TeX$ adds another macro called <code>\@sanitize</code> representing the same character set, but without the curly braces. The macros <code>\bbl@add@special⟨char⟩</code> and <code>\bbl@remove@special⟨char⟩</code> add and remove the character <code>⟨char⟩</code> to these two sets.

### 3.5 Support for saving macro definitions

Language definition files may want to *redefine* macros that already exist. Therefore a mechanism for saving (and restoring) the original definition of those macros is provided. We provide two macros for this<sup>27</sup>.

<code>\babel@save</code>	To save the current meaning of any control sequence, the macro <code>\babel@save</code> is provided. It takes one argument, <code>⟨csequence⟩</code> , the control sequence for which the meaning has to be saved.
<code>\babel@savevariable</code>	A second macro is provided to save the current value of a variable. In this context, anything that is allowed after the <code>\</code> the primitive is considered to be a variable. The macro takes one argument, the <code>⟨variable⟩</code> . The effect of the preceding macros is to append a piece of code to the current definition of <code>\originalTeX</code> . When <code>\originalTeX</code> 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

<code>\addto</code>	The macro <code>\addto{⟨control sequence⟩}{⟨<math>\TeX</math> code⟩}</code> can be used to extend the definition of a macro. The macro need not be defined (ie, it can be undefined or <code>\relax</code> ). This macro can, for instance, be used in adding instructions to a macro like <code>\extrasenglish</code> . 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 behavior is preserved for backward compatibility. If you are using <code>etoolbox</code> , by Philipp Lehman, consider using the tools provided by this package instead of <code>\addto</code> .
---------------------	--

### 3.7 Macros common to a number of languages

<code>\bbl@allowhyphens</code>	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
--------------------------------	--

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

allow hyphenation in the rest of such a compound word, the macro `\bbl@allowhyphens` can be used.

`\allowhyphens` Same as `\bbl@allowhyphens`, but does nothing if the encoding is T1. It is intended mainly for characters provided as real glyphs by this encoding but constructed with `\accent` in OT1.

Note the previous command (`\bbl@allowhyphens`) has different applications (hyphens and discretionaries) than this one (composite chars). Note also prior to version 3.7, `\allowhyphens` had the behavior of `\bbl@allowhyphens`.

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

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

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

### 3.8 Encoding-dependent strings

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

Furthermore, it must be activated explicitly, with the package option `strings`. If there is no `strings`, these blocks are ignored, except `\SetCases` (and except if forced as described 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 – a explicit name (or names) is much better and clearer. A “selector” is a name to be used as value in package option strings, optionally followed by extra info about the encodings to be used. The name `unicode` must be used for `xetex` and `luatex` (the key `strings` has also other two special values: `generic` and `encoded`). If a string is set several times (because several blocks are read), the first one takes precedence (ie, it works much like `\providecommand`).

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

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

Blocks without a selector are read always if the key strings has been used. They provide fallback values, and therefore must be the last blocks; they should be provided always if possible and all strings should be defined somehow inside it; they can be the only blocks (mainly LGC scripts using the LICR). Blocks without a selector can be activated explicitly with `strings=generic` (no block is taken into account except those). With `strings=encoded`, strings in those blocks are set as default (internally, ?). With `strings=encoded` strings are protected, but they are correctly expanded in `\MakeUppercase` and the like. If there is no key strings, string definitions are ignored, but `\SetCases` are still honoured (in a encoded way). The `<category>` is either captions, date or extras. You must stick to these three categories, even if no error is raised when using other name.<sup>28</sup> It may be empty, too, but in such a case using `\SetString` is an error (but not `\SetCase`).

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

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

\EndBabelCommands
```

A real example is:

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

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

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

```

\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).

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

The starred version just forces strings to take a value – if not set as package option, then the default for the engine is used. This is not done by default to prevent backward incompatibilities, but if you are creating a new language this version is better. It's up to the maintainers of the current languages to decide if using it is appropriate.<sup>29</sup>

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

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

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

**$\backslash SetString$**   $\{ \langle macro-name \rangle \} \{ \langle string \rangle \}$

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.

**$\backslash SetStringLoop$**   $\{ \langle macro-name \rangle \} \{ \langle string-list \rangle \}$

A convenient way to define several ordered names at once. For example, to define  $\backslash abmoniname$ ,  $\backslash abmoniiname$ , 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.

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

Sets globally code to be executed at  $\backslash MakeUpper case$  and  $\backslash MakeLower case$ . The code would be typically things like  $\backslash let \backslash BB \backslash bb$  and  $\backslash uccode$  or  $\backslash lccode$  (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  $\backslash @uc\lc list$  (eg,  $\backslash bb \backslash BB \backslash cc \backslash CC$ ). The mandatory arguments take precedence over the optional one. This command, unlike  $\backslash SetString$ , is executed always (even without strings), and it is intended for minor readjustments only. For example, as T1 is the default case mapping in  $\LaTeX$ , we could set for Turkish:

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



```

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

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

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

\EndBabelCommands

```

(Note the mapping for OT1 is not complete.)

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

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

```

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

```

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

## 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 behavior for shorthands across languages). These changes are described in this manual in the corresponding place. A selective list follows:

- `\select@language` did not set `\language`. This meant the language in force when auxiliary files were loaded was the one used in, for example, shorthands – if the language was german, a `\select@language{spanish}` had no effect.
- `\foreignlanguage` and `otherlanguage*` messed up `\extras<language>`. Scripts, encodings and many other things were not switched correctly.
- The `:ENC` mechanism for hyphenation patterns used the encoding of the *previous* language, not that of the language being selected.
- `'` (with `activeacute`) had the original value when writing to an auxiliary file, and things like an infinite loop 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 lead 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` 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  $\TeX$  package, which sets options and loads language styles.

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

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

## 6 locale directory

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

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

This is a preliminary documentation.

`ini` files contain the actual data; `tex` files are currently just proxies to the corresponding `ini` files.

Most keys are self-explanatory.

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

**version** of the `ini` file

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

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

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

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

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

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

## 7 Tools

```
1 <<version=3.22.1381>>
2 <<date=2018/08/27>>
```

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

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

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

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

`\bbl@afterelse` `\bbl@afterfi` Because the code that is used in the handling of active characters may need to look ahead, we take extra care to 'throw' it over the `\else` and `\fi` parts of an `\if`-statement<sup>30</sup>. These

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

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

```
26 \long\def\bbl@afterelse#1\else#2\fi{\fi#1}
27 \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.

```
28 \def\bbl@tempa#1{%
29   \long\def\bbl@trim##1##2{%
30     \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
31   \def\bbl@trim@c{%
32     \ifx\bbl@trim@a\@sptoken
33       \expandafter\bbl@trim@b
34     \else
35       \expandafter\bbl@trim@b\expandafter#1%
36     \fi}%
37   \long\def\bbl@trim@b#1##1 \@nil{\bbl@trim@i##1}}
38 \bbl@tempa{ }
39 \long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
40 \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.

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

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

```
59 \def\bbl@ifblank#1{%
60   \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
61 \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).

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

```

63 \def\bbl@kvcmd##1##2##3{#2}%
64 \bbl@kvnext#1,\@nil,}
65 \def\bbl@kvnext#1,{%
66 \ifx\@nil#1\relax\else
67 \bbl@ifblank{#1}{\bbl@forkv@eq#1=\@empty=\@nil{#1}}%
68 \expandafter\bbl@kvnext
69 \fi}
70 \def\bbl@forkv@eq#1=#2=#3\@nil#4{%
71 \bbl@trim@def\bbl@forkv@a{#1}%
72 \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).

```

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

```

**\bbl@replace**

```

82 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
83 \toks@{}}%
84 \def\bbl@replace@aux##1#2##2#2{%
85 \ifx\bbl@nil##2%
86 \toks@\expandafter{\the\toks@##1}%
87 \else
88 \toks@\expandafter{\the\toks@##1#3}%
89 \bbl@afterfi
90 \bbl@replace@aux##2#2%
91 \fi}%
92 \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
93 \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.

```

94 \def\bbl@exp#1{%
95 \begingroup
96 \let\ \noexpand
97 \def\<##1>\expandafter\noexpand\csname##1\endcsname}%
98 \edef\bbl@exp@aux{\endgroup#1}%
99 \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 pdf<sub>T</sub><sub>E</sub>X, 1 is luatex, and 2 is xetex. You may use the latter it in your language style if you want.

```

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

```

```

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

```

123 <<(*Make sure ProvidesFile is defined)>> ≡
124 \ifx\ProvidesFile\@undefined
125     \def\ProvidesFile#1[#2 #3 #4]{%
126         \wlog{File: #1 #4 #3 <#2>}%
127         \let\ProvidesFile\@undefined}
128 \fi
129 <</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`.

```

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

```

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

```

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

```

## 7.1 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.

```

142 <<(*Define core switching macros)>> ≡
143 \ifx\language\@undefined
144     \csname newcount\endcsname\language
145 \fi
146 <</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.

```

147 <<*Define core switching macros>> ≡
148 \ifx\newlanguage\undefined
149   \csname newcount\endcsname\last@language
150   \def\addlanguage#1{%
151     \global\advance\last@language\@ne
152     \ifnum\last@language<\@cclvi
153       \else
154         \errmessage{No room for a new \string\language!}%
155       \fi
156       \global\chardef#1\last@language
157       \wlog{\string#1 = \string\language\the\last@language}}
158   \else
159     \countdef\last@language=19
160     \def\addlanguage{\alloc@9\language\chardef\@cclvi}
161   \fi
162 <</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.

## 8 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 behavior of language definition files.

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



## 8.1 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.

```
163 (*package)
164 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
165 \ProvidesPackage{babel}[\langle\date\rangle\ \langle\version\rangle\ The Babel package]
166 \@ifpackagewith{babel}{debug}
167   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
168    \let\bbl@debug\@firstofone}
169   {\providecommand\bbl@trace[1]{}%
170    \let\bbl@debug\@gobble}
171 \ifx\bbl@switchflag\undefined % Prevent double input
172   \let\bbl@switchflag\relax
173   \input switch.def\relax
174 \fi
175 \langle\Load patterns in luatex\rangle
176 \langle\Basic macros\rangle
177 \def\AfterBabelLanguage#1{%
178   \global\expandafter\bbl@add\csname#1.ldf-h@k\endcsname}%
179 }
```

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.

```
179 \ifx\bbl@languages\undefined\else
180   \begingroup
181     \catcode\^^I=12
182     \@ifpackagewith{babel}{showlanguages}{%
183       \begingroup
184         \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
185         \wlog{<*languages>}%
186         \bbl@languages
187         \wlog{</languages>}%
188       \endgroup}{%
189     \endgroup
190     \def\bbl@elt#1#2#3#4{%
191       \ifnum#2=\z@
192         \gdef\bbl@nulllanguage{#1}%
193         \def\bbl@elt##1##2##3##4{}%
194       \fi}%
195     \bbl@languages
196 \fi
197 \ifodd\bbl@engine
198   \let\bbl@tempa\relax
199   \@ifpackagewith{babel}{bidi=basic}%
200   {\def\bbl@tempa{basic}}%
201   {\@ifpackagewith{babel}{bidi=basic-r}%
202    {\def\bbl@tempa{basic-r}}%
203   {}}
204 \ifx\bbl@tempa\relax\else
205   \let\bbl@beforeforeign\leavevmode
206   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}%
207   \RequirePackage{luatexbase}%
208   \directlua{
209     require('babel-bidi.lua')
210     require('babel-bidi-\bbl@tempa.lua')
211     luatexbase.add_to_callback('pre_linebreak_filter',
212       Babel.pre_otfload_v,
213       'Babel.pre_otfload_v',
214     }
215 }
```

```

214     luatexbase.priority_in_callback('pre_linebreak_filter',
215     'luaotfload.node_processor') or nil)
216   luatexbase.add_to_callback('hpack_filter',
217     Babel.pre_otfload_h,
218     'Babel.pre_otfload_h',
219     luatexbase.priority_in_callback('hpack_filter',
220     'luaotfload.node_processor') or nil)
221 }
222 \fi
223 \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.

```

224 \bbl@trace{Defining option 'base'}
225 \@ifpackagewith{babel}{base}{%
226   \ifx\directlua\@undefined
227     \DeclareOption*{\bbl@patterns{\CurrentOption}}%
228   \else
229     \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
230   \fi
231   \DeclareOption{base}{}%
232   \DeclareOption{showlanguages}{}%
233   \ProcessOptions
234   \global\expandafter\let\csname opt@babel.sty\endcsname\relax
235   \global\expandafter\let\csname ver@babel.sty\endcsname\relax
236   \global\let@ifl@ter@@\@ifl@ter
237   \def@ifl@ter#1#2#3#4#5{\global\let@ifl@ter\@ifl@ter@@}%
238   \endinput}{}%

```

## 8.2 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.

```

239 \bbl@trace{key=value and another general options}
240 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
241 \def\bbl@tempb#1.#2{%
242   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
243 \def\bbl@tempd#1.#2\@nnil{%
244   \ifx\@empty#2%
245     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
246   \else
247     \in@{=}{#1}\ifin@
248     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
249   \else
250     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
251   \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
252   \fi
253   \fi}
254 \let\bbl@tempc\@empty
255 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
256 \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.

```

257 \DeclareOption{KeepShorthandsActive}{}
258 \DeclareOption{activeacute}{}
259 \DeclareOption{activegrave}{}
260 \DeclareOption{debug}{}
261 \DeclareOption{noconfigs}{}
262 \DeclareOption{showlanguages}{}
263 \DeclareOption{silent}{}
264 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
265 <<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.

```

266 \let\bbl@opt@shorthands\@nnil
267 \let\bbl@opt@config\@nnil
268 \let\bbl@opt@main\@nnil
269 \let\bbl@opt@headfoot\@nnil
270 \let\bbl@opt@layout\@nnil

```

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

```

271 \def\bbl@tempa#1=#2\bbl@tempa{%
272   \bbl@csarg\ifx{opt@#1}\@nnil
273     \bbl@csarg\edef{opt@#1}{#2}%
274   \else
275     \bbl@error{%
276       Bad option `#1=#2'. Either you have misspelled the\\%
277       key or there is a previous setting of `#1'}{%
278       Valid keys are `shorthands', `config', `strings', `main',\\%
279       `headfoot', `safe', `math', among others.}
280   \fi}

```

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

```

281 \let\bbl@language@opts\@empty
282 \DeclareOption*{%
283   \bbl@xin@{\string=}{\CurrentOption}%
284   \ifin@
285     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
286   \else
287     \bbl@add@list\bbl@language@opts{\CurrentOption}%
288   \fi}

```

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

```

289 \ProcessOptions*

```

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

```

290 \bbl@trace{Conditional loading of shorthands}
291 \def\bbl@sh@string#1{%
292   \ifx#1\@empty\else
293     \ifx#1t\string~%
294     \else\ifx#1c\string,%
295     \else\string#1%
296   \fi\fi
297   \expandafter\bbl@sh@string
298 \fi}
299 \ifx\bbl@opt@shorthands\@nnil
300   \def\bbl@ifshorthand#1#2#3{#2}%
301 \else\ifx\bbl@opt@shorthands\@empty
302   \def\bbl@ifshorthand#1#2#3{#3}%
303 \else

```

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

```

304 \def\bbl@ifshorthand#1{%
305   \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
306   \ifin@
307     \expandafter\@firstoftwo
308   \else
309     \expandafter\@secondoftwo
310   \fi}

```

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

```

311 \edef\bbl@opt@shorthands{%
312   \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.

```

313 \bbl@ifshorthand{'}%
314   {\PassOptionsToPackage{activeacute}{babel}}{}
315 \bbl@ifshorthand{`}%
316   {\PassOptionsToPackage{activegrave}{babel}}{}
317 \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.

```

318 \ifx\bbl@opt@headfoot\@nnil\else
319   \g@addto@macro\@resetactivechars{%
320     \set@typeset@protect
321     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
322     \let\protect\noexpand}
323 \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.

```

324 \ifx\bbl@opt@safe\@undefined
325   \def\bbl@opt@safe{BR}
326 \fi
327 \ifx\bbl@opt@main\@nnil\else
328   \edef\bbl@language@opts{%
329     \ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
330     \bbl@opt@main}
331 \fi

```

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

```

332 \bbl@trace{Defining IfBabelLayout}
333 \ifx\bbl@opt@layout\@nnil
334 \newcommand\IfBabelLayout[3]{#3}%
335 \else
336 \newcommand\IfBabelLayout[1]{%
337 \expandafter\in@{.#1.}{.\bbl@opt@layout.}%
338 \ifin@
339 \expandafter\@firstoftwo
340 \else
341 \expandafter\@secondoftwo
342 \fi}
343 \fi

```

## 8.4 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).

```

344 \bbl@trace{Language options}
345 \let\bbl@afterlang\relax
346 \let\BabelModifiers\relax
347 \let\bbl@loaded\@empty
348 \def\bbl@load@language#1{%
349 \InputIfFileExists{#1.ldf}%
350 {\edef\bbl@loaded{\CurrentOption
351 \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
352 \expandafter\let\expandafter\bbl@afterlang
353 \csname\CurrentOption.ldf-h@@k\endcsname
354 \expandafter\let\expandafter\BabelModifiers
355 \csname bbl@mod@\CurrentOption\endcsname}%
356 {\bbl@error{%
357 Unknown option '\CurrentOption'. Either you misspelled it\\
358 or the language definition file \CurrentOption.ldf was not found}{%
359 Valid options are: shorthands=, KeepShorthandsActive,\\
360 activeacute, activegrave, noconfigs, safe=, main=, math=\\
361 headfoot=, strings=, config=, hyphenmap=, or a language name.}}}

```

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

```

362 \def\bbl@try@load@lang#1#2#3{%
363 \IfFileExists{\CurrentOption.ldf}%
364 {\bbl@load@language{\CurrentOption}}%
365 {#1\bbl@load@language{#2}#3}}
366 \DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}}
367 \DeclareOption{brazil}{\bbl@try@load@lang{}{portuges}}
368 \DeclareOption{brazilian}{\bbl@try@load@lang{}{portuges}}
369 \DeclareOption{hebrew}{%
370 \input{rlbabel.def}%
371 \bbl@load@language{hebrew}}
372 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}}
373 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}}
374 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}}
375 \DeclareOption{polutonikogreek}{%
376 \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
377 \DeclareOption{portuguese}{\bbl@try@load@lang{}{portuges}}
378 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}}

```

```

379 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
380 \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.

```

381 \ifx\bbl@opt@config\@nnil
382   \@ifpackagewith{babel}{noconfigs}{}%
383   {\InputIfFileExists{bblopts.cfg}%
384     {\typeout{*****^J%
385               * Local config file bblopts.cfg used^^J%
386               *}}%
387     {}}%
388 \else
389   \InputIfFileExists{\bbl@opt@config.cfg}%
390   {\typeout{*****^J%
391             * Local config file \bbl@opt@config.cfg used^^J%
392             *}}%
393   {\bbl@error{%
394     Local config file '\bbl@opt@config.cfg' not found}{%
395     Perhaps you misspelled it.}}%
396 \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.

```

397 \bbl@for\bbl@tempa\bbl@language@opts{%
398   \bbl@ifunset{ds@\bbl@tempa}%
399   {\edef\bbl@tempb{%
400     \noexpand\DeclareOption
401     {\bbl@tempa}%
402     {\noexpand\bbl@load@language{\bbl@tempa}}}%
403     \bbl@tempb}%
404   \@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.

```

405 \bbl@foreach\@classoptionslist{%
406   \bbl@ifunset{ds@#1}%
407   {\IfFileExists{#1.ldf}%
408     {\DeclareOption{#1}{\bbl@load@language{#1}}}%
409     {}}%
410   {}}

```

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

```

411 \ifx\bbl@opt@main\@nnil\else
412   \expandafter
413   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
414   \DeclareOption{\bbl@opt@main}{}
415 \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):

```
416 \def\AfterBabelLanguage#1{%
417   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
418 \DeclareOption*{}
419 \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`.

```
420 \ifx\bbl@opt@main\@nnil
421   \edef\bbl@tempa{\c@optionslist,\bbl@language@opts}
422   \let\bbl@tempc\@empty
423   \bbl@for\bbl@tempb\bbl@tempa{%
424     \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
425     \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
426   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
427   \expandafter\bbl@tempa\bbl@loaded,\@nnil
428   \ifx\bbl@tempb\bbl@tempc\else
429     \bbl@warning{%
430       Last declared language option is '\bbl@tempc',\%
431       but the last processed one was '\bbl@tempb'.\%
432       The main language cannot be set as both a global\%
433       and a package option. Use 'main=\bbl@tempc' as\%
434       option. Reported}%
435   \fi
436 \else
437   \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
438   \ExecuteOptions{\bbl@opt@main}
439   \DeclareOption*{}
440   \ProcessOptions*
441 \fi
442 \def\AfterBabelLanguage{%
443   \bbl@error
444   {Too late for \string\AfterBabelLanguage}%
445   {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.

```
446 \ifx\bbl@main@language\@undefined
447   \bbl@info{%
448     You haven't specified a language. I'll use 'nil'\%
449     as the main language. Reported}
450   \bbl@load@language{nil}
451 \fi
452 \</package>
453 \<core>
```

## 9 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  $\text{\LaTeX}$ -specific stuff. Because plain  $\text{\TeX}$  users might want to use some of the features of the babel system too, care has to be taken that plain  $\text{\TeX}$  can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain  $\text{\TeX}$  and  $\text{\LaTeX}$ , some of it is for the  $\text{\LaTeX}$  case only. Plain formats based on `etex` (`etex`, `xetex`, `luatex`) don’t load `hyphen.cfg` but `etex.src`, which follows a different naming convention, so we need to define the babel names. It presumes `language.def` exists and it is the same file used when formats were created.

## 9.1 Tools

```

454 \ifx\ldf@quit\@undefined
455 \else
456   \expandafter\endinput
457 \fi
458 <<Make sure ProvidesFile is defined>>
459 \ProvidesFile{babel.def}[<<date>> <<version>> Babel common definitions]
460 <<Load macros for plain if not LaTeX>>

```

The file `babel.def` expects some definitions made in the  $\text{\LaTeX} 2_{\epsilon}$  style file. So, In  $\text{\LaTeX} 2.09$  and Plain we must provide at least some predefined values as well some tools to set them (even if not all options are available). There in no package options, and therefore and 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).

```

461 \ifx\bbl@ifshorthand\@undefined
462   \let\bbl@opt@shorthands\@nnil
463   \def\bbl@ifshorthand#1#2#3{#2}%
464   \let\bbl@language@opts\@empty
465   \ifx\babeloptionstrings\@undefined
466     \let\bbl@opt@strings\@nnil
467   \else
468     \let\bbl@opt@strings\babeloptionstrings
469   \fi
470   \def\BabelStringsDefault{generic}
471   \def\bbl@tempa{normal}
472   \ifx\babeloptionmath\bbl@tempa
473     \def\bbl@mathnormal{\noexpand\textormath}
474   \fi
475   \def\AfterBabelLanguage#1#2{}
476   \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
477   \let\bbl@afterlang\relax
478   \def\bbl@opt@safe{BR}
479   \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
480   \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
481 \fi

```

And continue.

```

482 \ifx\bbl@switchflag\@undefined % Prevent double input
483   \let\bbl@switchflag\relax
484   \input switch.def\relax
485 \fi
486 \bbl@trace{Compatibility with language.def}
487 \ifx\bbl@languages\@undefined
488   \ifx\directlua\@undefined

```



```

489 \openin1 = language.def
490 \ifeof1
491 \closein1
492 \message{I couldn't find the file language.def}
493 \else
494 \closein1
495 \begingroup
496 \def\addlanguage#1#2#3#4#5{%
497 \expandafter\ifx\csname lang@#1\endcsname\relax\else
498 \global\expandafter\let\csname l@#1\expandafter\endcsname
499 \csname lang@#1\endcsname
500 \fi}%
501 \def\uselanguage#1{}%
502 \input language.def
503 \endgroup
504 \fi
505 \fi
506 \chardef\l@english\z@
507 \fi
508 <<Load patterns in luatex>>
509 <<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  $\TeX$ -code to be added to the *<control sequence>*.

If the *<control sequence>* has not been defined before it is defined now. The control sequence could also expand to `\relax`, in which case a circular definition results. The net result is a stack overflow. Otherwise the replacement text for the *<control sequence>* is expanded and stored in a token register, together with the  $\TeX$ -code to be added. Finally the *<control sequence>* is redefined, using the contents of the token register.

```

510 \def\addto#1#2{%
511 \ifx#1\@undefined
512 \def#1{#2}%
513 \else
514 \ifx#1\relax
515 \def#1{#2}%
516 \else
517 {\toks@\expandafter{#1#2}%
518 \xdef#1{\the\toks@}}%
519 \fi
520 \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.

```

521 \def\bbl@withactive#1#2{%
522 \begingroup
523 \lccode`~=`#2\relax
524 \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@. . .`

```

525 \def\bbl@redefine#1{%

```

```

526 \edef\bbl@tempa{\bbl@stripslash#1}%
527 \expandafter\let\csname org@\bbl@tempa\endcsname#1%
528 \expandafter\def\csname\bbl@tempa\endcsname}

```

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

```
529 \@onlypreamble\bbl@redefine
```

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

```

530 \def\bbl@redefine@long#1{%
531   \edef\bbl@tempa{\bbl@stripslash#1}%
532   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
533   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
534 \@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_`.

```

535 \def\bbl@redefineroobust#1{%
536   \edef\bbl@tempa{\bbl@stripslash#1}%
537   \bbl@ifunset{\bbl@tempa\space}%
538   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
539     \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
540   {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
541   \@namedef{\bbl@tempa\space}}

```

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

```
542 \@onlypreamble\bbl@redefineroobust
```

## 9.2 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.

```

543 \bbl@trace{Hooks}
544 \def\AddBabelHook#1#2{%
545   \bbl@ifunset{bbl@hk@#1}{\EnableBabelHook{#1}}}%
546   \def\bbl@tempa##1,#2=##2,##3\@empty{\def\bbl@tempb{##2}}%
547   \expandafter\bbl@tempa\bbl@evargs,#2=,\@empty
548   \bbl@ifunset{bbl@ev@#1@#2}%
549     {\bbl@csarg\bbl@add{ev@#2}{\bbl@elt{#1}}}%
550     \bbl@csarg\newcommand}%
551     {\bbl@csarg\let{ev@#1@#2}\relax
552     \bbl@csarg\newcommand}%
553     {ev@#1@#2}[\bbl@tempb]}
554 \def\EnableBabelHook#1{\bbl@csarg\let{hk@#1}\@firstofone}
555 \def\DisableBabelHook#1{\bbl@csarg\let{hk@#1}\@gobble}
556 \def\bbl@usehooks#1#2{%
557   \def\bbl@elt##1{%
558     \@nameuse{bbl@hk@##1}{\@nameuse{bbl@ev@##1@#1}#2}}%
559   \@nameuse{bbl@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).

```
560 \def\bbl@evargs{% <- don't delete this comma
561   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
562   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
563   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
564   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 `\bbl@e@<language>`. We register a hook at the `afterextras` event which just executes this macro in a “complete” selection (which, if undefined, is `\relax` and does nothing). This part is somewhat involved because we have to make sure things are expanded the correct number of times.

The macro `\bbl@e@<language>` contains `\bbl@ensure{<include>}{<exclude>}{<fontenc>}`, which in turn loops over the macros names in `\bbl@captionslist`, excluding (with the help of `\in@`) those in the exclude list. If the fontenc is given (and not `\relax`), the `\fontencoding` is also added. Then we loop over the include list, but if the macro already contains `\foreignlanguage`, nothing is done. Note this macro (1) is not restricted to the preamble, and (2) changes are local.

```
565 \bbl@trace{Defining babelensure}
566 \newcommand\babelensure[2][{}]{% TODO - revise test files
567   \AddBabelHook{babel-ensure}{afterextras}{%
568     \ifcase\bbl@select@type
569       \@nameuse{\bbl@e@\language\language}%
570     \fi}%
571   \begin{group}
572     \let\bbl@ens@include\@empty
573     \let\bbl@ens@exclude\@empty
574     \def\bbl@ens@fontenc{\relax}%
575     \def\bbl@tempb##1{%
576       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
577     \def\bbl@tempa{\bbl@tempb##1\@empty}%
578     \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ens@##1}{##2}}%
579     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
580     \def\bbl@tempc{\bbl@ensure}%
581     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
582       \expandafter{\bbl@ens@include}}%
583     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
584       \expandafter{\bbl@ens@exclude}}%
585     \toks@\expandafter{\bbl@tempc}%
586     \bbl@exp{%
587       \endgroup
588       \def<\bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
589     \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
590       \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
591         \ifx##1\@empty\else
592           \in@{##1}{#2}%
593         \ifin\@else
594           \bbl@ifunset{\bbl@ensure@\language\language}%
595             {\bbl@exp{%
596               \\DeclareRobustCommand\bbl@ensure@\language\language[1]{%
597                 \\foreignlanguage{\language\language}%
598                 {\ifx\relax#3\else
599                   \\fontencoding{#3}\\selectfont
600                   \fi
601                 #####1}}}%
602             }
```

```

602      {}%
603      \toks@\expandafter{##1}%
604      \edef##1{%
605        \bbl@csarg\noexpand{ensure@\language}%
606        {\the\toks@}}%
607      \fi
608      \expandafter\bbl@tempb
609    \fi}%
610  \expandafter\bbl@tempb\bbl@captionslist\today\@empty
611  \def\bbl@tempa##1{% elt for include list
612    \ifx##1\@empty\else
613      \bbl@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
614      \ifin\else
615        \bbl@tempb##1\@empty
616      \fi
617      \expandafter\bbl@tempa
618    \fi}%
619  \bbl@tempa#1\@empty}
620 \def\bbl@captionslist{%
621   \prefacename\refname\abstractname\bibname\chaptername\appendixname
622   \contentsname\listfigurename\listtablename\indexname\figurename
623   \tablename\partname\encname\ccname\headtoname\pagename\seename
624   \alsiname\proofname\glossaryname}

```

### 9.3 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`.

```

625 \bbl@trace{Macros for setting language files up}
626 \def\bbl@ldfinit{%
627   \let\bbl@screset\@empty
628   \let\BabelStrings\bbl@opt@string
629   \let\BabelOptions\@empty
630   \let\BabelLanguages\relax
631   \ifx\originalTeX\@undefined
632     \let\originalTeX\@empty
633   \else
634     \originalTeX

```

```

635 \fi}
636 \def\LdfInit#1#2{%
637 \chardef\atcatcode=\catcode`\@
638 \catcode`\@=11\relax
639 \chardef\eqcatcode=\catcode`\=
640 \catcode`\==12\relax
641 \expandafter\if\expandafter\@backslashchar
642 \expandafter\@car\string#2\@nil
643 \ifx#2\undefined\else
644 \ldf@quit{#1}%
645 \fi
646 \else
647 \expandafter\ifx\csname#2\endcsname\relax\else
648 \ldf@quit{#1}%
649 \fi
650 \fi
651 \bbl@ldfinit}

```

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

```

652 \def\ldf@quit#1{%
653 \expandafter\main@language\expandafter{#1}%
654 \catcode`\@=\atcatcode \let\atcatcode\relax
655 \catcode`\==\eqcatcode \let\eqcatcode\relax
656 \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.

```

657 \def\bbl@afterldf#1{%
658 \bbl@afterlang
659 \let\bbl@afterlang\relax
660 \let\BabelModifiers\relax
661 \let\bbl@screaset\relax}%
662 \def\ldf@finish#1{%
663 \loadlocalcfg{#1}%
664 \bbl@afterldf{#1}%
665 \expandafter\main@language\expandafter{#1}%
666 \catcode`\@=\atcatcode \let\atcatcode\relax
667 \catcode`\==\eqcatcode \let\eqcatcode\relax}

```

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

```

668 \@onlypreamble\LdfInit
669 \@onlypreamble\ldf@quit
670 \@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.

```

671 \def\main@language#1{%
672 \def\bbl@main@language{#1}%
673 \let\languagename\bbl@main@language
674 \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`.

```
675 \AtBeginDocument{%
676   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
677   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place
```

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

```
678 \def\select@language@x#1{%
679   \ifcase\bbl@select@type
680     \bbl@ifsamestring\languagename{#1}{\select@language{#1}}%
681   \else
682     \select@language{#1}%
683   \fi}
```

## 9.4 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.

```
684 \bbl@trace{Shorhands}
685 \def\bbl@add@special#1{% 1:a macro like "\, \?, etc.
686   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
687   \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\@makeother#1}}%
688   \ifx\nfss@catcodes\@undefined\else % TODO - same for above
689     \begingroup
690       \catcode`#1\active
691       \nfss@catcodes
692       \ifnum\catcode`#1=\active
693         \endgroup
694         \bbl@add\nfss@catcodes{\@makeother#1}%
695       \else
696         \endgroup
697       \fi
698   \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.

```
699 \def\bbl@remove@special#1{%
700   \begingroup
701     \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
702       \else\noexpand##1\noexpand##2\fi}%
703     \def\do{\x\do}%
704     \def\@makeother{\x\@makeother}%
705   \edef\x{\endgroup
706     \def\noexpand\dospecials{\dospecials}%
707     \expandafter\ifx\csgname @sanitize\endcsgname\relax\else
708       \def\noexpand\@sanitize{\@sanitize}%
709     \fi}%
710   \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).

```

711 \def\bbl@active@def#1#2#3#4{%
712   \@namedef{#3#1}{%
713     \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
714       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
715     \else
716       \bbl@afterfi\csname#2@sh@#1@\endcsname
717     \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.

```

718 \long\@namedef{#3@arg#1}##1{%
719   \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
720     \bbl@afterelse\csname#4#1\endcsname##1%
721   \else
722     \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
723   \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.

```

724 \def\initiate@active@char#1{%
725   \bbl@ifunset{active@char\string#1}%
726   {\bbl@withactive
727     {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
728   {}}

```

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`).

```

729 \def\@initiate@active@char#1#2#3{%
730   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
731   \ifx#1\@undefined
732     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
733   \else
734     \bbl@csarg\let{oridef@#2}#1%
735     \bbl@csarg\edef{oridef@#2}{%
736       \let\noexpand#1%
737       \expandafter\noexpand\csname\bbl@oridef@@#2\endcsname}%
738   \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`).

```

739 \ifx#1#3\relax
740   \expandafter\let\csname normal@char#2\endcsname#3%
741 \else
742   \bbl@info{Making #2 an active character}%
743   \ifnum\mathcode`#2="8000
744     \@namedef{normal@char#2}{%
745       \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
746   \else
747     \@namedef{normal@char#2}{#3}%
748   \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).

```

749   \bbl@restoreactive{#2}%
750   \AtBeginDocument{%
751     \catcode`#2\active
752     \if@filesw
753       \immediate\write\@mainaux{\catcode`\string#2\active}%
754     \fi}%
755   \expandafter\bbl@add@special\csname#2\endcsname
756   \catcode`#2\active
757 \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⟩`).

```

758 \let\bbl@tempa\@firstoftwo
759 \if\string^#2%
760   \def\bbl@tempa{\noexpand\textormath}%
761 \else
762   \ifx\bbl@mathnormal\@undefined\else
763     \let\bbl@tempa\bbl@mathnormal
764   \fi
765 \fi
766 \expandafter\edef\csname active@char#2\endcsname{%
767   \bbl@tempa
768     {\noexpand\if@safe@actives
769       \noexpand\expandafter
770       \expandafter\noexpand\csname normal@char#2\endcsname
771     \noexpand\else
772       \noexpand\expandafter
773       \expandafter\noexpand\csname bbl@doactive#2\endcsname
774     \noexpand\fi}%
775   {\expandafter\noexpand\csname normal@char#2\endcsname}}%

```



```

776 \bbl@csarg\edef{doactive#2}{%
777   \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

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

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

```

778 \bbl@csarg\edef{active@#2}{%
779   \noexpand\active@prefix\noexpand#1%
780   \expandafter\noexpand\csname active@char#2\endcsname}%
781 \bbl@csarg\edef{normal@#2}{%
782   \noexpand\active@prefix\noexpand#1%
783   \expandafter\noexpand\csname normal@char#2\endcsname}%
784 \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.

```

785 \bbl@active@def#2\user@group{user@active}{language@active}%
786 \bbl@active@def#2\language@group{language@active}{system@active}%
787 \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  $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  would see  $\backslash\mathrm{protect}'\backslash\mathrm{protect}'$ . To prevent this from happening a couple of shorthand needs to be defined at user level.

```

788 \expandafter\edef\csname\user@group @sh@#2@@\endcsname
789   {\expandafter\noexpand\csname normal@char#2\endcsname}%
790 \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
791   {\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  $\backslash\mathrm{pr@m@s}$  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.

```

792 \if\string'#2%
793   \let\prim@s\bbl@prim@s
794   \let\active@math@prime#1%
795 \fi
796 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}

```

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

```

797 <<(*More package options)>> \equiv
798 \DeclareOption{math=active}{}
799 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
800 <</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*.

```

801 \@ifpackagewith{babel}{KeepShorthandsActive}%
802   {\let\bbl@restoreactive@gobble}%
803   {\def\bbl@restoreactive#1{%
804     \bbl@exp{%

```

```

805      \\AfterBabelLanguage\\CurrentOption
806      {\catcode`#1=\the\catcode`#1\relax}%
807      \\AtEndOfPackage
808      {\catcode`#1=\the\catcode`#1\relax}}}%
809      \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.

```

810 \def\bbl@sh@select#1#2{%
811   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
812     \bbl@afterelse\bbl@scndcs
813   \else
814     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
815   \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`.

```

816 \def\active@prefix#1{%
817   \ifx\protect\@typeset@protect
818     \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).

```

819     \ifx\protect\@unexpandable@protect
820       \noexpand#1%
821     \else
822       \protect#1%
823     \fi
824     \expandafter\@gobble
825   \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⟩`.

```

826 \newif\if@safe@actives
827 \@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.

```

828 \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`.

```

829 \def\bbl@activate#1{%
830   \bbl@withactive{\expandafter\let\expandafter}#1%
831   \csname bbl@active@\string#1\endcsname}
832 \def\bbl@deactivate#1{%

```

```

833 \bbl@withactive{\expandafter\let\expandafter}#1%
834 \csname bbl@normal@\string#1\endcsname}

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

835 \def\bbl@firstcs#1#2{\csname#1\endcsname}
836 \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.

837 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
838 \def\@decl@short#1#2#3\@nil#4{%
839 \def\bbl@tempa{#3}%
840 \ifx\bbl@tempa\@empty
841 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
842 \bbl@ifunset{#1@sh@\string#2@}{}%
843 {\def\bbl@tempa{#4}%
844 \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
845 \else
846 \bbl@info
847 {Redefining #1 shorthand \string#2\\%
848 in language \CurrentOption}%
849 \fi}%
850 \@namedef{#1@sh@\string#2@}{#4}%
851 \else
852 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
853 \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
854 {\def\bbl@tempa{#4}%
855 \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
856 \else
857 \bbl@info
858 {Redefining #1 shorthand \string#2\string#3\\%
859 in language \CurrentOption}%
860 \fi}%
861 \@namedef{#1@sh@\string#2@\string#3@}{#4}%
862 \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.

863 \def\textormath{%
864 \ifmmode
865 \expandafter\@secondoftwo
866 \else
867 \expandafter\@firstoftwo
868 \fi}

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

869 \def\user@group{user}
870 \def\language@group{english}
871 \def\system@group{system}

```

`\useshortands` 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.

```

872 \def\useshortands{%
873   \@ifstar\bb1@usesesh@s{\bb1@usesesh@x{}}
874 \def\bb1@usesesh@s#1{%
875   \bb1@usesesh@x
876   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb1@activate{#1}}}%
877   {#1}}
878 \def\bb1@usesesh@x#1#2{%
879   \bb1@ifshorthand{#2}%
880   {\def\user@group{user}%
881     \initiate@active@char{#2}%
882     #1%
883     \bb1@activate{#2}}%
884   {\bb1@error
885     {Cannot declare a shorthand turned off (\string#2)}
886     {Sorry, but you cannot use shorthands which have been\\%
887       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.

```

888 \def\user@language@group{user@\language@group}
889 \def\bb1@set@user@generic#1#2{%
890   \bb1@ifunset{user@generic@active#1}%
891   {\bb1@active@def#1\user@language@group{user@active}{user@generic@active}%
892     \bb1@active@def#1\user@group{user@generic@active}{language@active}%
893     \expandafter\edef\csname#2@sh@#1@@\endcsname{%
894       \expandafter\noexpand\csname normal@char#1\endcsname}%
895     \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%
896       \expandafter\noexpand\csname user@active#1\endcsname}}%
897   \@empty}
898 \newcommand\defineshorthand[3][user]{%
899   \edef\bb1@tempa{\zap@space#1 \@empty}%
900   \bb1@for\bb1@tempb\bb1@tempa{%
901     \if*\expandafter\car\bb1@tempb\@nil
902       \edef\bb1@tempb{user\expandafter@gobble\bb1@tempb}%
903       \@expandtwoargs
904       \bb1@set@user@generic{\expandafter\string\car#2\@nil}\bb1@tempb
905     \fi
906     \declare@shorthand{\bb1@tempb}{#2}{#3}}}

```

`\languageshortands` 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.

```

907 \def\languageshortands#1{\def\language@group{#1}}

```

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

```

908 \def\aliasshorthand#1#2{%
909   \bb1@ifshorthand{#2}%
910   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
911     \ifx\document\@notprerr

```

```

912      \@notshorthand{#2}%
913      \else
914      \initiate@active@char{#2}%

```

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

```

915      \expandafter\let\csname active@char\string#2\expandafter\endcsname
916      \csname active@char\string#1\endcsname
917      \expandafter\let\csname normal@char\string#2\expandafter\endcsname
918      \csname normal@char\string#1\endcsname
919      \bbl@activate{#2}%
920      \fi
921    \fi}%
922  {\bbl@error
923    {Cannot declare a shorthand turned off (\string#2)}
924    {Sorry, but you cannot use shorthands which have been\\%
925      turned off in the package options}}}

```

`\@notshorthand`

```

926 \def\@notshorthand#1{%
927   \bbl@error{%
928     The character '\string #1' should be made a shorthand character;\\%
929     add the command \string\usesshorthands\string{#1\string} to
930     the preamble.\\%
931     I will ignore your instruction}%
932   {You may proceed, but expect unexpected results}}

```

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

`\shorthandoff`

```

933 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
934 \DeclareRobustCommand*\shorthandoff{%
935   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
936 \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.

```

937 \def\bbl@switch@sh#1#2{%
938   \ifx#2\@nnil\else
939     \bbl@ifunset{\bbl@active@\string#2}%
940     {\bbl@error
941       {I cannot switch '\string#2' on or off--not a shorthand}%
942       {This character is not a shorthand. Maybe you made\\%
943         a typing mistake? I will ignore your instruction}}%
944     {\ifcase#1%
945       \catcode`#2\relax
946       \or
947       \catcode`#2\active
948       \or
949       \csname bbl@oricat@\string#2\endcsname

```

```

950      \csname bbl@oridef@string#2\endcsname
951      \fi}%
952      \bbl@afterfi\bbl@switch@sh#1%
953      \fi}

```

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

```

954 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
955 \def\bbl@putsh#1{%
956   \bbl@ifunset{\bbl@active@string#1}%
957   {\bbl@putsh@i#1\empty\@nnil}%
958   {\csname bbl@active@string#1\endcsname}}
959 \def\bbl@putsh@i#1#2\@nnil{%
960   \csname\language @sh@string#1@%
961     \ifx\empty#2\else\string#2\fi\endcsname}
962 \ifx\bbl@opt@shorthands\@nnil\else
963   \let\bbl@s@initiate@active@char\initiate@active@char
964   \def\initiate@active@char#1{%
965     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
966   \let\bbl@s@switch@sh\bbl@switch@sh
967   \def\bbl@switch@sh#1#2{%
968     \ifx#2\@nnil\else
969       \bbl@afterfi
970       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
971       \fi}
972   \let\bbl@s@activate\bbl@activate
973   \def\bbl@activate#1{%
974     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
975   \let\bbl@s@deactivate\bbl@deactivate
976   \def\bbl@deactivate#1{%
977     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
978   \fi

```

You may want to test if a character is a shorthand. Mainly intended for packages, and therefore marked with camel case. Note it does not test whether the shorthand is on or off.

```

979 \def\IfBabelShorthand#1#2#3{\bbl@ifunset{\bbl@active@string#1}{#3}{#2}}

```

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

```

980 \def\bbl@prim@s{%
981   \prime\futurelet\@let@token\bbl@pr@m@s}
982 \def\bbl@if@primes#1#2{%
983   \ifx#1\@let@token
984     \expandafter\@firstoftwo
985   \else\ifx#2\@let@token
986     \bbl@afterelse\expandafter\@firstoftwo
987   \else
988     \bbl@afterfi\expandafter\@secondoftwo
989   \fi\fi}
990 \begingroup
991   \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
992   \catcode`\'=12 \catcode`\"=\active \lccode`\"=\'
993   \lowercase{%
994     \gdef\bbl@pr@m@s{%
995       \bbl@if@primes"%
996       \pr@@@s

```

```

997      {\bbl@if@primes*^{\pr@@@t\egroup}}
998 \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).

```

999 \initiate@active@char{~}
1000 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1001 \bbl@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.

```

1002 \expandafter\def\csname OT1dqpos\endcsname{127}
1003 \expandafter\def\csname T1dqpos\endcsname{4}

```

When the macro `\f@encoding` is undefined (as it is in plain  $\TeX$ ) we define it here to expand to OT1

```

1004 \ifx\f@encoding\@undefined
1005   \def\f@encoding{OT1}
1006 \fi

```

## 9.5 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.

```

1007 \bbl@trace{Language attributes}
1008 \newcommand\languageattribute[2]{%
1009   \def\bbl@tempc{#1}%
1010   \bbl@fixname\bbl@tempc
1011   \bbl@iflanguage\bbl@tempc{%
1012     \bbl@vforeach{#2}{%

```

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

```

1013     \ifx\bbl@known@attribs\@undefined
1014       \in@false
1015     \else

```

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

```

1016       \bbl@xin@{\bbl@tempc-##1,}{\bbl@known@attribs,}%
1017     \fi

```

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

```

1018     \ifin@
1019       \bbl@warning{%
1020         You have more than once selected the attribute '##1'\%
1021         for language #1. Reported}%
1022     \else

```

When we end up here the attribute is not selected before. So, we add it to the list of selected attributes and execute the associated  $\TeX$ -code.

```

1023      \bbl@exp{%
1024        \\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
1025      \edef\bbl@tempa{\bbl@tempc-##1}%
1026      \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1027      {\csname\bbl@tempc @attr@##1\endcsname}%
1028      {\@attrerr{\bbl@tempc}{##1}}%
1029      \fi}}

```

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

```
1030 \@onlypreamble\languageattribute
```

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

```

1031 \newcommand*{\@attrerr}[2]{%
1032   \bbl@error
1033   {The attribute #2 is unknown for language #1.}%
1034   {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}`.

```

1035 \def\bbl@declare@ttribute#1#2#3{%
1036   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1037   \ifin@
1038     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1039   \fi
1040   \bbl@add@list\bbl@attributes{#1-#2}%
1041   \expandafter\def\csname#1@attr@#2\endcsname{#3}}

```

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

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

```
1042 \def\bbl@ifattributeset#1#2#3#4{%
```

First we need to find out if any attributes were set; if not we're done.

```

1043   \ifx\bbl@known@attribs\@undefined
1044     \in@false
1045     \else

```

The we need to check the list of known attributes.

```

1046     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1047     \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`'.

```

1048   \ifin@
1049     \bbl@afterelse#3%
1050   \else
1051     \bbl@afterfi#4%
1052   \fi
1053 }

```



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

```
1054 \def\bbl@ifknown@ttrib#1#2{%
```

We first assume the attribute is unknown.

```
1055 \let\bbl@tempa\@secondoftwo
```

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

```
1056 \bbl@loopx\bbl@tempb{#2}{%
```

```
1057 \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{, #1,}%
```

```
1058 \ifin@
```

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

```
1059 \let\bbl@tempa\@firstoftwo
```

```
1060 \else
```

```
1061 \fi}%
```

Finally we execute `\bbl@tempa`.

```
1062 \bbl@tempa
```

```
1063 }
```

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

```
1064 \def\bbl@clear@ttribs{%
```

```
1065 \ifx\bbl@attributes\undefined\else
```

```
1066 \bbl@loopx\bbl@tempa{\bbl@attributes}{%
```

```
1067 \expandafter\bbl@clear@ttrib\bbl@tempa.
```

```
1068 }%
```

```
1069 \let\bbl@attributes\undefined
```

```
1070 \fi}
```

```
1071 \def\bbl@clear@ttrib#1-#2.{%
```

```
1072 \expandafter\let\csname#1@attr#2\endcsname\undefined}
```

```
1073 \AtBeginDocument{\bbl@clear@ttribs}
```

## 9.6 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 1074 \bbl@trace{Macros for saving definitions}
1075 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1076 \newcount\babel@savecnt
```

```
1077 \babel@beginsave
```

`\babel@save` The macro `\babel@save<csname>` saves the current meaning of the control sequence `<csname>` to `\originalTeX`<sup>31</sup>. To do this, we let the current meaning to a temporary control

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

sequence, the restore commands are appended to `\originalTeX` and the counter is incremented.

```
1078 \def\babel@save#1{%
1079   \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
1080   \toks@\expandafter{\originalTeX\let#1=}%
1081   \bbl@exp{%
1082     \def\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1083   \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.

```
1084 \def\babel@savevariable#1{%
1085   \toks@\expandafter{\originalTeX #1}%
1086   \bbl@exp{\def\originalTeX{\the\toks@the#1\relax}}}
```

`\bbl@frenchspacing` Some languages need to have `\frenchspacing` in effect. Others don't want that. The command `\bbl@frenchspacing` switches it on when it isn't already in effect and `\bbl@nonfrenchspacing` switches it off if necessary.

```
1087 \def\bbl@frenchspacing{%
1088   \ifnum\the\sfcodes\@m
1089     \let\bbl@nonfrenchspacing\relax
1090   \else
1091     \frenchspacing
1092     \let\bbl@nonfrenchspacing\nonfrenchspacing
1093   \fi}
1094 \let\bbl@nonfrenchspacing\nonfrenchspacing
```

## 9.7 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 `\csname` but the actual macro.

```
1095 \bbl@trace{Short tags}
1096 \def\babeltags#1{%
1097   \edef\bbl@tempa{\zap@space#1 \@empty}%
1098   \def\bbl@tempb##1##2\@{#}%
1099   \edef\bbl@tempc{%
1100     \noexpand\newcommand
1101     \expandafter\noexpand\csname ##1\endcsname{%
1102       \noexpand\protect
1103       \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1104     \noexpand\newcommand
1105     \expandafter\noexpand\csname text##1\endcsname{%
1106       \noexpand\foreignlanguage{##2}}
1107   \bbl@tempc}%
1108   \bbl@for\bbl@tempa\bbl@tempa{%
1109     \expandafter\bbl@tempb\bbl@tempa\@{#}}
```

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

```
1110 \bbl@trace{Hyphens}
```

```

1111 \@onlypreamble\babelhyphenation
1112 \AtEndOfPackage{%
1113   \newcommand\babelhyphenation[2][\@empty]{%
1114     \ifx\bbbl@hyphenation@relax
1115       \let\bbbl@hyphenation@\@empty
1116     \fi
1117     \ifx\bbbl@hyphlist@\@empty\else
1118       \bbbl@warning{%
1119         You must not intermingle \string\selectlanguage\space and\\%
1120         \string\babelhyphenation\space or some exceptions will not\\%
1121         be taken into account. Reported}%
1122     \fi
1123     \ifx\@empty#1%
1124       \protected@edef\bbbl@hyphenation@{\bbbl@hyphenation@\space#2}%
1125     \else
1126       \bbbl@vforeach{#1}{%
1127         \def\bbbl@tempa{##1}%
1128         \bbbl@fixname\bbbl@tempa
1129         \bbbl@iflanguage\bbbl@tempa{%
1130           \bbbl@csarg\protected@edef{hyphenation@\bbbl@tempa}{%
1131             \bbbl@ifunset{bbbl@hyphenation@\bbbl@tempa}%
1132             \@empty
1133             {\csname bbl@hyphenation@\bbbl@tempa\endcsname\space}%
1134             #2}}}%
1135       \fi}}

```

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

```

1136 \def\bbbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1137 \def\bbbl@t@one{T1}
1138 \def\allowhyphens{\ifx\cf@encoding\bbbl@t@one\else\bbbl@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`.

```

1139 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1140 \def\babelhyphen{\active@prefix\babelhyphen\bbbl@hyphen}
1141 \def\bbbl@hyphen{%
1142   \@ifstar{\bbbl@hyphen@i @}{\bbbl@hyphen@i \@empty}}
1143 \def\bbbl@hyphen@i#1#2{%
1144   \bbbl@ifunset{bbbl@hy#1#2\@empty}%
1145   {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{-}{#2}}}%
1146   {\csname bbl@hy#1#2\@empty\endcsname}}

```

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

```

1147 \def\bbbl@usehyphen#1{%
1148   \leavevmode
1149   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1150   \nobreak\hskip\z@skip}
1151 \def\bbbl@@usehyphen#1{%
1152   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

<sup>32</sup> $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

The following macro inserts the hyphen char.

```

1153 \def\bbl@hyphenchar{%
1154   \ifnum\hyphenchar\font=\m@ne
1155     \babeinullhyphen
1156   \else
1157     \char\hyphenchar\font
1158   \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@nbreak` is redundant.

```

1159 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1160 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1161 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1162 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1163 \def\bbl@hy@nbreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1164 \def\bbl@hy@nbreak{\mbox{\bbl@hyphenchar}}
1165 \def\bbl@hy@repeat{%
1166   \bbl@usehyphen{%
1167     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1168 \def\bbl@hy@repeat{%
1169   \bbl@usehyphen{%
1170     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1171 \def\bbl@hy@empty{\hskip\z@skip}
1172 \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.

```

1173 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#1}\bbl@allowhyphens}

```

## 9.9 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.

```

1174 \bbl@trace{Multiencoding strings}
1175 \def\bbl@tglobal#1{\global\let#1#1}
1176 \def\bbl@recatcode#1{%
1177   \@tempcnta="7F
1178   \def\bbl@tempa{%
1179     \ifnum\@tempcnta>"FF\else
1180       \catcode\@tempcnta=#1\relax
1181       \advance\@tempcnta\@ne
1182       \expandafter\bbl@tempa
1183     \fi}%
1184   \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 `\(lang)\bbl@uclc` because we do not know how many expansions are necessary (depends on whether strings are encoded). The last part is tricky – when uppercasing, we have:

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

and starts over (and similarly when lowercasing).

```
1185 \@ifpackagewith{babel}{nocase}%
1186 {\let\bbl@patchucl\relax}%
1187 {\def\bbl@patchucl{%
1188   \global\let\bbl@patchucl\relax
1189   \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1190   \gdef\bbl@uclc##1{%
1191     \let\bbl@encoded\bbl@encoded@uclc
1192     \bbl@ifunset{\language @bbl@uclc}% and resumes it
1193     {##1}%
1194     {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1195      \csname\language @bbl@uclc\endcsname}%
1196      {\bbl@tolower\@empty}{\bbl@toupper\@empty}}}%
1197   \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
1198   \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}%
1199 <<(*More package options)>> ≡
1200 \DeclareOption{nocase}{}
1201 <</More package options>>
```

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

```
1202 <<(*More package options)>> ≡
1203 \let\bbl@opt@strings\@nnil % accept strings=value
1204 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1205 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1206 \def\BabelStringsDefault{generic}
1207 <</More package options>>
```

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

```
1208 \@onlypreamble\StartBabelCommands
1209 \def\StartBabelCommands{%
1210   \begingroup
1211   \bbl@recatcode{11}%
1212   <<Macros local to BabelCommands>>
1213   \def\bbl@provstring##1##2{%
1214     \providecommand##1{##2}%
1215     \bbl@tglobal##1}%
1216   \global\let\bbl@scafter\@empty
1217   \let\StartBabelCommands\bbl@startcmds
1218   \ifx\BabelLanguages\relax
1219     \let\BabelLanguages\CurrentOption
1220   \fi
1221   \begingroup
1222   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1223   \StartBabelCommands}
1224 \def\bbl@startcmds{%
1225   \ifx\bbl@screset\@nnil\else
1226     \bbl@usehooks{stopcommands}{}%
1227   \fi
1228   \endgroup
1229   \begingroup
1230   \@ifstar
1231   {\ifx\bbl@opt@strings\@nnil
```

```

1232 \let\bbl@opt@strings\BabelStringsDefault
1233 \fi
1234 \bbl@startcmds@i}%
1235 \bbl@startcmds@i}
1236 \def\bbl@startcmds@i#1#2{%
1237 \edef\bbl@L{\zap@space#1 \@empty}%
1238 \edef\bbl@G{\zap@space#2 \@empty}%
1239 \bbl@startcmds@ii}

```

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

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

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

```

1240 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1241 \let\SetString@gobbletwo
1242 \let\bbl@stringdef@gobbletwo
1243 \let\AfterBabelCommands@gobble
1244 \ifx\@empty#1%
1245 \def\bbl@sc@label{generic}%
1246 \def\bbl@encstring##1##2{%
1247 \ProvideTextCommandDefault##1{##2}%
1248 \bbl@tglobal##1%
1249 \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%
1250 \let\bbl@sctest\in@true
1251 \else
1252 \let\bbl@sc@charset\space % <- zapped below
1253 \let\bbl@sc@fontenc\space % <- " "
1254 \def\bbl@tempa##1=##2\@nil{%
1255 \bbl@csarg\edef{sc\zap@space##1 \@empty}{##2 }}%
1256 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1257 \def\bbl@tempa##1 ##2{% space -> comma
1258 ##1%
1259 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1260 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1261 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1262 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1263 \def\bbl@encstring##1##2{%
1264 \bbl@foreach\bbl@sc@fontenc{%
1265 \bbl@ifunset{T@####1}%
1266 {}%
1267 {\ProvideTextCommand##1{####1}{##2}%
1268 \bbl@tglobal##1%
1269 \expandafter
1270 \bbl@tglobal\csname####1\string##1\endcsname}}}%
1271 \def\bbl@sctest{%
1272 \bbl@xin@{\bbl@opt@strings,}{\bbl@sc@label,\bbl@sc@fontenc,}}%
1273 \fi
1274 \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
1275 \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
1276 \let\AfterBabelCommands\bbl@aftercmds
1277 \let\SetString\bbl@setstring
1278 \let\bbl@stringdef\bbl@encstring

```

```

1279 \else % ie, strings=value
1280 \bbl@sctest
1281 \ifin@
1282 \let\AfterBabelCommands\bbl@aftercmds
1283 \let\SetString\bbl@setstring
1284 \let\bbl@stringdef\bbl@provstring
1285 \fi\fi\fi
1286 \bbl@scswitch
1287 \ifx\bbl@G\@empty
1288 \def\SetString##1##2{%
1289 \bbl@error{Missing group for string \string##1}%
1290 {You must assign strings to some category, typically\\%
1291 captions or extras, but you set none}}%
1292 \fi
1293 \ifx\@empty#1%
1294 \bbl@usehooks{defaultcommands}{}%
1295 \else
1296 \@expandtwoargs
1297 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1298 \fi}

```

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

```

1299 \def\bbl@forlang#1#2{%
1300 \bbl@for#1\bbl@L{%
1301 \bbl@xin@{,#1},{, \BabelLanguages,}%
1302 \ifin@#2\relax\fi}}
1303 \def\bbl@scswitch{%
1304 \bbl@forlang\bbl@tempa{%
1305 \ifx\bbl@G\@empty\else
1306 \ifx\SetString\@gobbletwo\else
1307 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1308 \bbl@xin@{,\bbl@GL,},{, \bbl@screset,}%
1309 \ifin@\else
1310 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1311 \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1312 \fi
1313 \fi
1314 \fi}}
1315 \AtEndOfPackage{%
1316 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{#{2}}}%
1317 \let\bbl@scswitch\relax}
1318 \@onlypreamble\EndBabelCommands
1319 \def\EndBabelCommands{%
1320 \bbl@usehooks{stopcommands}{}%
1321 \endgroup
1322 \endgroup
1323 \bbl@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.

```

1324 \def\bbl@setstring#1#2{%
1325   \bbl@forlang\bbl@tempa{%
1326     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1327     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1328     {\global\expandafter % TODO - con \bbl@exp ?
1329       \bbl@add\csname\bbl@G\bbl@tempa\expandafter\endcsname\expandafter
1330       {\expandafter\bbl@scset\expandafter#1\csname\bbl@LC\endcsname}}}%
1331     }%
1332   \def\BabelString{#2}%
1333   \bbl@usehooks{stringprocess}{}%
1334   \expandafter\bbl@stringdef
1335     \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`.

```

1336 \ifx\bbl@opt@strings\relax
1337   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1338   \bbl@patchuclc
1339   \let\bbl@encoded\relax
1340   \def\bbl@encoded@uclc#1{%
1341     \@inmathwarn#1%
1342     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1343       \expandafter\ifx\csname ?\string#1\endcsname\relax
1344         \TextSymbolUnavailable#1%
1345       \else
1346         \csname ?\string#1\endcsname
1347       \fi
1348     \else
1349       \csname\cf@encoding\string#1\endcsname
1350     \fi}
1351 \else
1352   \def\bbl@scset#1#2{\def#1{#2}}
1353 \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.

```

1354 <<*Macros local to BabelCommands>> ≡
1355 \def\SetStringLoop##1##2{%
1356   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1357   \count@\z@
1358   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1359     \advance\count@\@ne
1360     \toks@\expandafter{\bbl@tempa}%
1361     \bbl@exp{%
1362       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1363       \count@=\the\count@\relax}}}%
1364 <</Macros local to BabelCommands>>

```

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



```

1365 \def\bbl@aftercmds#1{%
1366   \toks@\expandafter{\bbl@scafter#1}%
1367   \xdef\bbl@scafter{\the\toks@}}

```

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

```

1368 <<(*Macros local to BabelCommands)>> ≡
1369   \newcommand\SetCase[3][]{%
1370     \bbl@patchuclc
1371     \bbl@forlang\bbl@tempa{%
1372       \expandafter\bbl@encstring
1373       \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
1374       \expandafter\bbl@encstring
1375       \csname\bbl@tempa @bbl@uc\endcsname{##2}%
1376       \expandafter\bbl@encstring
1377       \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
1378 <</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.

```

1379 <<(*Macros local to BabelCommands)>> ≡
1380   \newcommand\SetHyphenMap[1]{%
1381     \bbl@forlang\bbl@tempa{%
1382       \expandafter\bbl@stringdef
1383       \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1384 <</Macros local to BabelCommands>>

```

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

```

1385 \newcommand\BabelLower[2]{% one to one.
1386   \ifnum\lccode#1=#2\else
1387     \babel@savevariable{\lccode#1}%
1388     \lccode#1=#2\relax
1389   \fi}
1390 \newcommand\BabelLowerMM[4]{% many-to-many
1391   \@tempcnta=#1\relax
1392   \@tempcntb=#4\relax
1393   \def\bbl@tempa{%
1394     \ifnum\@tempcnta>#2\else
1395       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1396       \advance\@tempcnta#3\relax
1397       \advance\@tempcntb#3\relax
1398       \expandafter\bbl@tempa
1399     \fi}%
1400   \bbl@tempa}
1401 \newcommand\BabelLowerMO[4]{% many-to-one
1402   \@tempcnta=#1\relax
1403   \def\bbl@tempa{%
1404     \ifnum\@tempcnta>#2\else
1405       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1406       \advance\@tempcnta#3
1407       \expandafter\bbl@tempa
1408     \fi}%
1409   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1410 <<(*More package options)>> ≡

```

```

1411 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1412 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\ne@}
1413 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1414 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1415 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1416 <</More package options>>

```

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

```

1417 \AtEndOfPackage{%
1418   \ifx\bbl@opt@hyphenmap\undefined
1419     \bbl@xin{,}{\bbl@language@opts}%
1420     \chardef\bbl@opt@hyphenmap\ifin4\else\ne\fi
1421   \fi}

```

## 9.10 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.

```

1422 \bbl@trace{Macros related to glyphs}
1423 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
1424   \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
1425   \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.

```

1426 \def\save@sf@q#1{\leavevmode
1427   \begingroup
1428   \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
1429   \endgroup}

```

## 9.11 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`.

### 9.11.1 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.

```

1430 \ProvideTextCommand{\quotedblbase}{OT1}{%
1431   \save@sf@q{\set@low@box{\textquotedblright\}}%
1432   \box\z@\kern-.04em\bbl@allowhyphens}}

```

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

```

1433 \ProvideTextCommandDefault{\quotedblbase}{%
1434   \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

1435 \ProvideTextCommand{\quotesinglbase}{OT1}{%
1436   \save@sf@q{\set@low@box{\textquoteright\}}%
1437   \box\z@\kern-.04em\bbl@allowhyphens}}

```

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

```

1438 \ProvideTextCommandDefault{\quotesinglbase}{%
1439   \UseTextSymbol{OT1}{\quotesinglbase}}

```

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

```
\guillemotright 1440 \ProvideTextCommand{\guillemotleft}{OT1}{%
1441   \ifmmode
1442     \ll
1443   \else
1444     \save@sf@q{\nobreak
1445       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
1446   \fi}
1447 \ProvideTextCommand{\guillemotright}{OT1}{%
1448   \ifmmode
1449     \gg
1450   \else
1451     \save@sf@q{\nobreak
1452       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
1453   \fi}
```

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

```
1454 \ProvideTextCommandDefault{\guillemotleft}{%
1455   \UseTextSymbol{OT1}{\guillemotleft}}
1456 \ProvideTextCommandDefault{\guillemotright}{%
1457   \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
\guilsinglright 1458 \ProvideTextCommand{\guilsinglleft}{OT1}{%
1459   \ifmmode
1460     <%
1461   \else
1462     \save@sf@q{\nobreak
1463       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
1464   \fi}
1465 \ProvideTextCommand{\guilsinglright}{OT1}{%
1466   \ifmmode
1467     >%
1468   \else
1469     \save@sf@q{\nobreak
1470       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
1471   \fi}
```

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

```
1472 \ProvideTextCommandDefault{\guilsinglleft}{%
1473   \UseTextSymbol{OT1}{\guilsinglleft}}
1474 \ProvideTextCommandDefault{\guilsinglright}{%
1475   \UseTextSymbol{OT1}{\guilsinglright}}
```

### 9.11.2 Letters

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

```
1476 \DeclareTextCommand{\ij}{OT1}{%
1477   i\kern-0.02em\bbl@allowhyphens j}
1478 \DeclareTextCommand{\IJ}{OT1}{%
1479   I\kern-0.02em\bbl@allowhyphens J}
1480 \DeclareTextCommand{\ij}{T1}{\char188}
1481 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
1482 \ProvideTextCommandDefault{\ij}{%
1483   \UseTextSymbol{OT1}{\ij}}
1484 \ProvideTextCommandDefault{\IJ}{%
1485   \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).

```
1486 \def\crrtic@{\hrule height0.1ex width0.3em}
1487 \def\crttic@{\hrule height0.1ex width0.33em}
1488 \def\ddj@{%
1489   \setbox0\hbox{d}\dimen@=\ht0
1490   \advance\dimen@1ex
1491   \dimen@.45\dimen@
1492   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1493   \advance\dimen@ii.5ex
1494   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
1495 \def\DDJ@{%
1496   \setbox0\hbox{D}\dimen@=.55\ht0
1497   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1498   \advance\dimen@ii.15ex % correction for the dash position
1499   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
1500   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
1501   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
1502 %
1503 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
1504 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
1505 \ProvideTextCommandDefault{\dj}{%
1506   \UseTextSymbol{OT1}{\dj}}
1507 \ProvideTextCommandDefault{\DJ}{%
1508   \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.

```
1509 \DeclareTextCommand{\SS}{OT1}{SS}
1510 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

### 9.11.3 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 1511 \ProvideTextCommandDefault{\glq}{%
1512   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```

1513 \ProvideTextCommand{\grq}{T1}{%
1514   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
1515 \ProvideTextCommand{\grq}{TU}{%
1516   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
1517 \ProvideTextCommand{\grq}{OT1}{%
1518   \save@sf@q{\kern-.0125em
1519     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
1520     \kern.07em\relax}}
1521 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}

```

`\glqq` The ‘german’ double quotes.

```

\grqq 1522 \ProvideTextCommandDefault{\glqq}{%
1523   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

1524 \ProvideTextCommand{\grqq}{T1}{%
1525   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1526 \ProvideTextCommand{\grqq}{TU}{%
1527   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1528 \ProvideTextCommand{\grqq}{OT1}{%
1529   \save@sf@q{\kern-.07em
1530     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
1531     \kern.07em\relax}}
1532 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}

```

`\flq` The ‘french’ single guillemets.

```

\frq 1533 \ProvideTextCommandDefault{\flq}{%
1534   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
1535 \ProvideTextCommandDefault{\frq}{%
1536   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

`\flqq` The ‘french’ double guillemets.

```

\frqq 1537 \ProvideTextCommandDefault{\flqq}{%
1538   \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
1539 \ProvideTextCommandDefault{\frqq}{%
1540   \textormath{\guillemotright}{\mbox{\guillemotright}}}

```

#### 9.11.4 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  
`\umlautlow` positioning, the default will be `\umlauthigh` (the normal positioning).

```

1541 \def\umlauthigh{%
1542   \def\bb1@umlauta##1{\leavevmode\bggroup%
1543     \expandafter\accent\csname\fontencoding dqpos\endcsname
1544     ##1\bb1@allowhyphens\egroup}%
1545   \let\bb1@umlaute\bb1@umlauta}
1546 \def\umlautlow{%
1547   \def\bb1@umlauta{\protect\lower@umlaut}}
1548 \def\umlautelow{%
1549   \def\bb1@umlaute{\protect\lower@umlaut}}
1550 \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.

```
1551 \expandafter\ifx\csname U@D\endcsname\relax
1552   \csname newdimen\endcsname\U@D
1553 \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.

```
1554 \def\lower@umlaut#1{%
1555   \leavevmode\bgroup
1556     \U@D 1ex%
1557     {\setbox\z@\hbox{%
1558       \expandafter\char\csname\fontencoding dqpos\endcsname}%
1559       \dimen@ -.45ex\advance\dimen@\ht\z@
1560       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
1561     \expandafter\accent\csname\fontencoding dqpos\endcsname
1562     \fontdimen5\font\U@D #1%
1563   \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).

```
1564 \AtBeginDocument{%
1565   \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
1566   \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
1567   \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{i}}%
1568   \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{i}}%
1569   \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
1570   \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
1571   \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
1572   \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
1573   \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
1574   \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
1575   \DeclareTextCompositeCommand{\"}{OT1}{U}{\bbl@umlauta{U}}%
1576 }
```

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

```
1577 \ifx\l@english\undefined
1578   \chardef\l@english\z@
1579 \fi
1580 \main@language{english}
```

## 9.12 Layout

**Work in progress.**

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

```

1581 \bbl@trace{Bidi layout}
1582 \providecommand\IfBabelLayout[3]{#3}%
1583 \newcommand\BabelPatchSection[1]{%
1584   \@ifundefined{#1}{}{%
1585     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
1586     \@namedef{#1}{%
1587       \ifstar{\bbl@presec@s{#1}}%
1588       {\@dblarg{\bbl@presec@x{#1}}}}}%
1589 \def\bbl@presec@x#1[#2]#3{%
1590   \bbl@exp{%
1591     \select@language@x{\bbl@main@language}%
1592     \\\@nameuse{bbl@sspre@#1}%
1593     \\\@nameuse{bbl@ss@#1}%
1594     [\\foreignlanguage{\language}{\unexpanded{#2}}}%
1595     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
1596     \select@language@x{\language}}}%
1597 \def\bbl@presec@s#1#2{%
1598   \bbl@exp{%
1599     \select@language@x{\bbl@main@language}%
1600     \\\@nameuse{bbl@sspre@#1}%
1601     \\\@nameuse{bbl@ss@#1}*%
1602     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
1603     \select@language@x{\language}}}%
1604 \IfBabelLayout{sectioning}%
1605   {\BabelPatchSection{part}%
1606    \BabelPatchSection{chapter}%
1607    \BabelPatchSection{section}%
1608    \BabelPatchSection{subsection}%
1609    \BabelPatchSection{subsubsection}%
1610    \BabelPatchSection{paragraph}%
1611    \BabelPatchSection{subparagraph}}%
1612   \def\babel@toc#1{%
1613     \select@language@x{\bbl@main@language}}}%
1614 \IfBabelLayout{captions}%
1615   {\BabelPatchSection{caption}}}%

```

Now we load definition files for engines.

```

1616 \bbl@trace{Input engine specific macros}
1617 \ifcase\bbl@engine
1618   \input txtbabel.def
1619 \or
1620   \input luababel.def
1621 \or
1622   \input xebabel.def
1623 \fi

```

## 9.13 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.

```

1624 \bbl@trace{Creating languages and reading ini files}
1625 \newcommand\babelprovide[2][{}]{%
1626   \let\bbl@savelangname\language

```

```

1627 \def\languagename{#2}%
1628 \let\bbl@KVP@captions\@nil
1629 \let\bbl@KVP@import\@nil
1630 \let\bbl@KVP@main\@nil
1631 \let\bbl@KVP@script\@nil
1632 \let\bbl@KVP@language\@nil
1633 \let\bbl@KVP@dir\@nil
1634 \let\bbl@KVP@hyphenrules\@nil
1635 \let\bbl@KVP@mapfont\@nil
1636 \let\bbl@KVP@maparabic\@nil
1637 \bbl@forkv{#1}{\bbl@csarg\def{KVP@##1}{##2}}% TODO - error handling
1638 \ifx\bbl@KVP@import\@nil\else
1639   \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
1640   {\begingroup
1641     \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
1642     \InputIfFileExists{babel-#2.tex}{}}%
1643   \endgroup}%
1644   {}%
1645 \fi
1646 \ifx\bbl@KVP@captions\@nil
1647   \let\bbl@KVP@captions\bbl@KVP@import
1648 \fi
1649 \bbl@ifunset{date#2}%
1650   {\bbl@provide@new{#2}}%
1651   {\bbl@ifblank{#1}%
1652     {\bbl@error
1653       {If you want to modify `#2' you must tell how in\\
1654       the optional argument. Currently there are three\\
1655       options: captions=lang-tag, hyphenrules=lang-list\\
1656       import=lang-tag}%
1657       {Use this macro as documented}}%
1658     {\bbl@provide@renew{#2}}}%
1659 \bbl@exp{\bbl@babelensure[exclude=\today]{#2}}%
1660 \bbl@ifunset{\bbl@ensure@\languagename}%
1661   {\bbl@exp{%
1662     \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
1663       \\\foreignlanguage{\languagename}%
1664       {###1}}}%
1665   }%
1666 \ifx\bbl@KVP@script\@nil\else
1667   \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
1668 \fi
1669 \ifx\bbl@KVP@language\@nil\else
1670   \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
1671 \fi
1672 \ifx\bbl@KVP@mapfont\@nil\else
1673   \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}}%
1674   {\bbl@error{Option `bbl@KVP@mapfont' unknown for\\
1675     mapfont. Use `direction'.%
1676     {See the manual for details.}}}%
1677   \bbl@ifunset{\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}}%
1678   \bbl@ifunset{\bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}}%
1679 \ifx\bbl@mapselect\@undefined
1680   \AtBeginDocument{%
1681     \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
1682     {\selectfont}}%
1683   \def\bbl@mapselect{%
1684     \let\bbl@mapselect\relax
1685     \edef\bbl@prefontid{\fontid\font}}%

```





```

1742 \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
1743 \@namedef{extras#1}{}%
1744 \@namedef{noextras#1}{}%
1745 \StartBabelCommands*{#1}{captions}%
1746   \ifx\bbbl@KVP@captions\@nil %      and also if import, implicit
1747   \def\bbbl@tempb##1{%              elt for \bbbl@captionslist
1748     \ifx##1\@empty\else
1749       \bbbl@exp{%
1750         \\SetString\\##1{%
1751           \\bbbl@nocaption{\bbbl@stripslash##1}{#1\bbbl@stripslash##1}}}%
1752       \expandafter\bbbl@tempb
1753     \fi}%
1754   \expandafter\bbbl@tempb\bbbl@captionslist\@empty
1755   \else
1756     \bbbl@read@ini{\bbbl@KVP@captions}% Here all letters cat = 11
1757     \bbbl@after@ini
1758     \bbbl@savestrings
1759   \fi
1760 \StartBabelCommands*{#1}{date}%
1761   \ifx\bbbl@KVP@import\@nil
1762     \bbbl@exp{%
1763       \\SetString\\today{\\bbbl@nocaption{today}{#1today}}}%
1764   \else
1765     \bbbl@savetoday
1766     \bbbl@savedate
1767   \fi
1768 \EndBabelCommands
1769 \bbbl@exp{%
1770   \def\<#1hyphenmins>{%
1771     {\bbbl@ifunset{\bbbl@lfthm@#1}{2}{\@nameuse{\bbbl@lfthm@#1}}}%
1772     {\bbbl@ifunset{\bbbl@rgthm@#1}{3}{\@nameuse{\bbbl@rgthm@#1}}}}}%
1773 \bbbl@provide@hyphens{#1}%
1774 \ifx\bbbl@KVP@main\@nil\else
1775   \expandafter\main@language\expandafter{#1}%
1776 \fi}
1777 \def\bbbl@provide@renew#1{%
1778   \ifx\bbbl@KVP@captions\@nil\else
1779     \StartBabelCommands*{#1}{captions}%
1780     \bbbl@read@ini{\bbbl@KVP@captions}% Here all letters cat = 11
1781     \bbbl@after@ini
1782     \bbbl@savestrings
1783   \EndBabelCommands
1784 \fi
1785 \ifx\bbbl@KVP@import\@nil\else
1786   \StartBabelCommands*{#1}{date}%
1787   \bbbl@savetoday
1788   \bbbl@savedate
1789   \EndBabelCommands
1790 \fi
1791 \bbbl@provide@hyphens{#1}}

```

The hyphenrules option is handled with an auxiliary macro.

```

1792 \def\bbbl@provide@hyphens#1{%
1793   \let\bbbl@tempa\relax
1794   \ifx\bbbl@KVP@hyphenrules\@nil\else
1795     \bbbl@replace\bbbl@KVP@hyphenrules{ }{,}%
1796     \bbbl@foreach\bbbl@KVP@hyphenrules{%
1797       \ifx\bbbl@tempa\relax % if not yet found
1798         \bbbl@ifsamestring{##1}{+}%

```

```

1799      {\bbl@exp{\addlanguage\<l@##1>}}}%
1800      {}%
1801      \bbl@ifunset{l@##1}%
1802      {}%
1803      {\bbl@exp{\let\bbl@tempa\<l@##1>}}%
1804      \fi}%
1805  \fi
1806  \ifx\bbl@tempa\relax %      if no opt or no language in opt found
1807    \ifx\bbl@KVP@import\@nil\else % if importing
1808      \bbl@exp{%      and hyphenrules is not empty
1809        \bbl@ifblank{\@nameuse{bbl@hyphr@#1}}}%
1810        {}%
1811        {\let\bbl@tempa\<l@\@nameuse{bbl@hyphr@language}>}}%
1812      \fi
1813      \fi
1814      \bbl@ifunset{bbl@tempa}%      ie, relax or undefined
1815      {\bbl@ifunset{l@#1}%      no hyphenrules found - fallback
1816        {\bbl@exp{\adddialect\<l@#1>\language}}}%
1817        {}}%      so, l@<lang> is ok - nothing to do
1818        {\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.*

```

1819 \def\bbl@read@ini#1{%
1820   \openin1=babel-#1.ini
1821   \ifeof1
1822     \bbl@error
1823     {There is no ini file for the requested language\%
1824      (#1). Perhaps you misspelled it or your installation\%
1825      is not complete.}%
1826     {Fix the name or reinstall babel.}%
1827   \else
1828     \let\bbl@section\@empty
1829     \let\bbl@savestrings\@empty
1830     \let\bbl@savetoday\@empty
1831     \let\bbl@savestate\@empty
1832     \let\bbl@inireader\bbl@iniskip
1833     \bbl@info{Importing data from babel-#1.ini for \language}%
1834     \loop
1835     \if T\ifeof1F\fi T\relax % Trick, because inside \loop
1836       \endlinechar\m@ne
1837       \read1 to \bbl@line
1838       \endlinechar\^^M
1839       \ifx\bbl@line\@empty\else
1840         \expandafter\bbl@inline\bbl@line\bbl@inline
1841       \fi
1842     \repeat
1843   \fi}
1844 \def\bbl@inline#1\bbl@inline{%
1845   \@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.

```

1846 \def\bbl@iniskip#1\@{}%      if starts with ;
1847 \def\bbl@inisec[#1]#2\@{}%   if starts with opening bracket
1848   \@nameuse{bbl@secpost\bbl@section}% ends previous section
1849 \def\bbl@section{#1}%
1850   \@nameuse{bbl@secpres\bbl@section}% starts current section

```

```

1851 \bbl@ifunset{bbl@secline@#1}%
1852 {\let\bbl@inireader\bbl@iniskip}%
1853 {\bbl@exp{\let\\bbl@inireader\<bbl@secline@#1>}}

```

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

```

1854 \def\bbl@inikv#1=#2\@@{%      key=value
1855 \bbl@trim@def\bbl@tempa{#1}%
1856 \bbl@trim\toks@{#2}%
1857 \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.

```

1858 \def\bbl@exportkey#1#2#3{%
1859 \bbl@ifunset{bbl@kv@#2}%
1860 {\bbl@csarg\gdef{#1@\language\name}{#3}}%
1861 {\expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
1862 \bbl@csarg\gdef{#1@\language\name}{#3}}%
1863 \else
1864 \bbl@exp{\global\let\<bbl@#1@\language\name>\<bbl@kv@#2>}%
1865 \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.

```

1866 \let\bbl@secline@identification\bbl@inikv
1867 \def\bbl@secpost@identification{%
1868 \bbl@exportkey{lname}{identification.name.english}{}%
1869 \bbl@exportkey{lbcpr}{identification.tag.bcp47}{}%
1870 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
1871 \bbl@exportkey{sname}{identification.script.name}{}%
1872 \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
1873 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}}
1874 \let\bbl@secline@typography\bbl@inikv
1875 \let\bbl@secline@numbers\bbl@inikv
1876 \def\bbl@after@ini{%
1877 \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
1878 \bbl@exportkey{rgtm}{typography.righthyphenmin}{3}%
1879 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
1880 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
1881 \bbl@xin@{0.5}{\@nameuse{bbl@kv@identification.version}}%
1882 \ifin@
1883 \bbl@warning{%
1884 There are neither captions nor date in '\language\name'.\%
1885 It may not be suitable for proper typesetting, and it\%
1886 could change. Reported}%
1887 \fi
1888 \bbl@xin@{0.9}{\@nameuse{bbl@kv@identification.version}}%
1889 \ifin@
1890 \bbl@warning{%
1891 The '\language\name' date format may not be suitable\%
1892 for proper typesetting, and therefore it very likely will\%
1893 change in a future release. Reported}%
1894 \fi
1895 \bbl@toglobal\bbl@savetoday
1896 \bbl@toglobal\bbl@savetoday

```

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

```

1897 \ifcase\bbl@engine
1898 \bbl@csarg\def{secline@captions.licr}{#1=#2\@@{%

```

```

1899 \bbl@ini@captions@aux{#1}{#2}}
1900 \bbl@csarg\def{secline@date.gregorian}#1=#2\@@{% for defaults
1901 \bbl@ini@dategreg#1...\relax{#2}}
1902 \bbl@csarg\def{secline@date.gregorian.licr}#1=#2\@@{% override
1903 \bbl@ini@dategreg#1...\relax{#2}}
1904 \else
1905 \def\bbl@secline@captions#1=#2\@@{%
1906 \bbl@ini@captions@aux{#1}{#2}}
1907 \bbl@csarg\def{secline@date.gregorian}#1=#2\@@{%
1908 \bbl@ini@dategreg#1...\relax{#2}}
1909 \fi

```

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

```

1910 \def\bbl@ini@captions@aux#1#2{%
1911 \bbl@trim@def\bbl@tempa{#1}%
1912 \bbl@ifblank{#2}%
1913 {\bbl@exp{%
1914 \toks@{\bbl@nocaption{\bbl@tempa}{\language\language\bbl@tempa name}}}%
1915 {\bbl@trim\toks@{#2}}%
1916 \bbl@exp{%
1917 \bbl@add\bbl@savestrings{%
1918 \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.

```

1919 \bbl@csarg\def{secpref@date.gregorian.licr}{%
1920 \ifcase\bbl@engine\let\bbl@savestate\empty\fi}
1921 \def\bbl@ini@dategreg#1.#2.#3.#4\relax#5% TODO - ignore with 'captions'
1922 \bbl@trim@def\bbl@tempa{#1.#2}%
1923 \bbl@ifsamestring{\bbl@tempa}{months.wide}%
1924 {\bbl@trim@def\bbl@tempa{#3}%
1925 \bbl@trim\toks@{#5}%
1926 \bbl@exp{%
1927 \bbl@add\bbl@savestate{%
1928 \SetString\<month\romannumeral\bbl@tempa name>{\the\toks@}}}%
1929 {\bbl@ifsamestring{\bbl@tempa}{date.long}%
1930 {\bbl@trim@def\bbl@toreplace{#5}%
1931 \bbl@TG@date
1932 \global\bbl@csarg\let{date@\language}\bbl@toreplace
1933 \bbl@exp{%
1934 \gdef\<\language date>{\protect\<\language date >}%
1935 \gdef\<\language date >####1####2####3}%
1936 \bbl@usedategroupttrue
1937 \<\bbl@ensure@\language>{%
1938 \<\bbl@date@\language>{####1}{####2}{####3}}}%
1939 \bbl@add\bbl@savetoday{%
1940 \SetString\<\today>%
1941 \<\language date>{\the\year}{\the\month}{\the\day}}}%
1942 {}

```

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.

```

1943 \newcommand\BabelDateSpace{\nobreakspace}
1944 \newcommand\BabelDateDot{.\@}
1945 \newcommand\BabelDated[1]{\number#1}
1946 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
1947 \newcommand\BabelDateM[1]{\number#1}
1948 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}

```

```

1949 \newcommand\BabelDateMMMM[1]{\%
1950 \csname month\romannumeral#1name\endcsname}%
1951 \newcommand\BabelDatey[1]{\number#1}%
1952 \newcommand\BabelDateyy[1]{\%
1953 \ifnum#1<10 0\number#1 %
1954 \else\ifnum#1<100 \number#1 %
1955 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
1956 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
1957 \else
1958 \bbl@error
1959 {Currently two-digit years are restricted to the\
1960 range 0-9999.}%
1961 {There is little you can do. Sorry.}%
1962 \fi\fi\fi\fi}}
1963 \newcommand\BabelDateyyyy[1]{\number#1}}
1964 \def\bbl@replace@finish@iii#1{\%
1965 \bbl@exp{\def\#1####1####2####3{\the\toks@}}%
1966 \def\bbl@TG@date{\%
1967 \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
1968 \bbl@replace\bbl@toreplace{[. ]}{\BabelDateDot{}}%
1969 \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
1970 \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
1971 \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
1972 \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
1973 \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
1974 \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
1975 \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
1976 \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
1977 % Note after \bbl@replace \toks@ contains the resulting string.
1978 % TODO - Using this implicit behavior doesn't seem a good idea.
1979 \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.

```

1980 \def\bbl@provide@lsys#1{\%
1981 \bbl@ifunset{bbl@lname@#1}%
1982 {\bbl@ini@ids{#1}}%
1983 {}%
1984 \bbl@csarg\let{lsys@#1}\@empty
1985 \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}}%
1986 \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}}%
1987 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
1988 \bbl@ifunset{bbl@lname@#1}{\%
1989 {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
1990 \bbl@csarg\bbl@tglobal{lsys@#1}}%
1991 \bbl@exp{\% TODO - should be global, but even local does its job
1992 % I'm still not sure -- must investigate
1993 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cs{sname@#1}}%
1994 {\newfontscript{\bbl@cs{sname@#1}}{\bbl@cs{sotf@#1}}}%
1995 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cs{lname@#1}}%
1996 {\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.

```

1997 \def\bbl@ini@ids#1{\%
1998 \def\BabelBeforeIni##1##2{\%
1999 \begingroup

```

```

2000      \bbl@add\bbl@secpost@identification{\closein1 }%
2001      \catcode`\[=12 \catcode`\]=12 \catcode`\==12 %
2002      \bbl@read@ini{##1}%
2003      \endgroup}%          boxed, to avoid extra spaces:
2004      {\setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}}{}}}

```

## 10 The kernel of Babel (babel.def, only $\text{\LaTeX}$ )

### 10.1 The redefinition of the style commands

The rest of the code in this file can only be processed by  $\text{\LaTeX}$ , so we check the current format. If it is plain  $\text{\TeX}$ , 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  $\text{\TeX}$  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`.

```

2005 {\def\format{lpplain}
2006 \ifx\fmtname\format
2007 \else
2008   \def\format{LaTeX2e}
2009   \ifx\fmtname\format
2010   \else
2011     \aftergroup\endinput
2012   \fi
2013 \fi}

```

### 10.2 Cross referencing macros

The  $\text{\LaTeX}$  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  $\text{\TeX}$ book [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.

```

2014 %\bbl@redefine\newlabel#1#2{%
2015 %  \@safe@activestrue\org@newlabel{#1}{#2}\@safe@activesfalse}

```

`\@newl@bel` We need to change the definition of the  $\text{\LaTeX}$ -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.

```

2016 <<(*More package options)>> ≡
2017 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}

```

```

2018 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
2019 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
2020 <</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.

```

2021 \bbl@trace{Cross referencing macros}
2022 \ifx\bbl@opt@safe\empty\else
2023   \def\@newl@bel#1#2#3{%
2024     {\@safe@activetrue
2025       \bbl@ifunset{#1@#2}%
2026       \relax
2027       {\gdef\@multiplelabels{%
2028         \@latex@warning@no@line{There were multiply-defined labels}}}%
2029       \@latex@warning@no@line{Label `#2' multiply defined}}}%
2030   \global\@namedef{#1@#2}{#3}}

```

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

```

2031 \CheckCommand*\@testdef[3]{%
2032   \def\reserved@a{#3}%
2033   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
2034   \else
2035     \@tempwattrue
2036   \fi}

```

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

```

2037 \def\@testdef#1#2#3{%
2038   \@safe@activetrue

```

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

```

2039   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname

```

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

```

2040   \def\bbl@tempb{#3}%
2041   \@safe@activesfalse

```

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

```

2042   \ifx\bbl@tempa\relax
2043   \else
2044     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
2045   \fi

```

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

```

2046   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%

```

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

```

2047   \ifx\bbl@tempa\bbl@tempb
2048   \else
2049     \@tempwattrue
2050   \fi}
2051 \fi

```



`\ref`    The same holds for the macro `\ref` that references a label and `\pageref` to reference a  
`\pageref` 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.

```
2052 \bbl@xin@{R}\bbl@opt@safe
2053 \ifin@
2054   \bbl@redefineroobust\ref#1{%
2055     \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
2056   \bbl@redefineroobust\pageref#1{%
2057     \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
2058 \else
2059   \let\org@ref\ref
2060   \let\org@pageref\pageref
2061 \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.

```
2062 \bbl@xin@{B}\bbl@opt@safe
2063 \ifin@
2064   \bbl@redefine\@citex[#1]#2{%
2065     \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
2066     \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.

```
2067   \AtBeginDocument{%
2068     \@ifpackageloaded{natbib}{%
```

Notice that we use `\def` here instead of `\bbl@redefine` because `\org@@citex` is already defined and we don't want to overwrite that definition (it would result in parameter stack overflow because of a circular definition).

(Recent versions of `natbib` change dynamically `\@citex`, so PR4087 doesn't seem fixable in a simple way. Just load `natbib` before.)

```
2069     \def\@citex[#1][#2]#3{%
2070       \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
2071       \org@@citex[#1][#2]{\@tempa}}%
2072   }{}}
```

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

```
2073   \AtBeginDocument{%
2074     \@ifpackageloaded{cite}{%
2075       \def\@citex[#1]#2{%
2076         \@safe@activetrue\org@@citex[#1][#2]\@safe@activesfalse}%
2077       }{}}
```

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

```
2078   \bbl@redefine\nocite#1{%
2079     \@safe@activetrue\org@nocite{#1}\@safe@activesfalse}
```

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

```
2080 \bbl@redefine\bibcite{%
```

We call `\bbl@cite@choice` to select the proper definition for `\bibcite`. This new definition is then activated.

```
2081 \bbl@cite@choice
```

```
2082 \bibcite}
```

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

```
2083 \def\bbl@bibcite#1#2{%
```

```
2084 \org@bibcite{#1}{\@safe@activesfalse#2}}
```

`\bbl@cite@choice` The macro `\bbl@cite@choice` determines which definition of `\bibcite` is needed.

```
2085 \def\bbl@cite@choice{%
```

First we give `\bibcite` its default definition.

```
2086 \global\let\bibcite\bbl@bibcite
```

Then, when `natbib` is loaded we restore the original definition of `\bibcite`.

```
2087 \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
```

For `cite` we do the same.

```
2088 \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
```

Make sure this only happens once.

```
2089 \global\let\bbl@cite@choice\relax}
```

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

```
2090 \AtBeginDocument{\bbl@cite@choice}
```

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

```
2091 \bbl@redefine\@bibitem#1{%
```

```
2092 \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
```

```
2093 \else
```

```
2094 \let\org@nocite\nocite
```

```
2095 \let\org@@citex\citex
```

```
2096 \let\org@bibcite\bibcite
```

```
2097 \let\org@@bibitem\@bibitem
```

```
2098 \fi
```

### 10.3 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@activetrue` is in effect.

```

2099 \bbl@trace{Marks}
2100 \IfBabelLayout{sectioning}
2101   {\ifx\bbl@opt@headfoot\@nnil
2102     \g@addto@macro\@resetactivechars{%
2103       \set@typeset@protect
2104       \expandafter\select@language@x\expandafter{\bbl@main@language}%
2105       \let\protect\noexpand
2106       \edef\thepage{%
2107         \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
2108     \fi}
2109   {\bbl@redefine\markright#1{%
2110     \bbl@ifblank{#1}%
2111     {\org@markright{}}%
2112     {\toks@{#1}%
2113      \bbl@exp{%
2114        \\\org@markright{\protect\\foreignlanguage{\language}%
2115          {\protect\\bbl@restore@actives\the\toks@}}}%

```

`\markboth`    The definition of `\markboth` is equivalent to that of `\markright`, except that we need two  
`\@mkboth`    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`.

```

2116   \ifx\@mkboth\markboth
2117     \def\bbl@tempc{\let\@mkboth\markboth}
2118   \else
2119     \def\bbl@tempc{}
2120   \fi

```

Now we can start the new definition of `\markboth`

```

2121   \bbl@redefine\markboth#1#2{%
2122     \protected@edef\bbl@tempb##1{%
2123       \protect\foreignlanguage
2124       {\language}\protect\bbl@restore@actives##1}%
2125     \bbl@ifblank{#1}%
2126     {\toks@{}}%
2127     {\toks@\expandafter{\bbl@tempb{#1}}}%
2128     \bbl@ifblank{#2}%
2129     {\@temptokena{}}%
2130     {\@temptokena\expandafter{\bbl@tempb{#2}}}%
2131     \bbl@exp{\@org@markboth{\the\toks@}{\the\@temptokena}}%

```

and copy it to `\@mkboth` if necessary.

```

2132   \bbl@tempc} % end \IfBabelLayout

```

## 10.4 Preventing clashes with other packages

### 10.4.1 `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.

```

2133 \bbl@trace{Preventing clashes with other packages}
2134 \bbl@xin@{R}\bbl@opt@safe
2135 \ifin@
2136   \AtBeginDocument{%
2137     \ifpackageloaded{ifthen}{%

```

Then we can redefine `\ifthenelse`:

```

2138       \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.

```

2139         \let\bbl@temp@pref\pageref
2140         \let\pageref\org@pageref
2141         \let\bbl@temp@ref\ref
2142         \let\ref\org@ref

```

Then we can set the `\@safe@activestrue` 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.

```

2143         \@safe@activestrue
2144         \org@ifthenelse{#1}%
2145         {\let\pageref\bbl@temp@pref
2146          \let\ref\bbl@temp@ref
2147          \@safe@activestrue
2148          #2}%
2149         {\let\pageref\bbl@temp@pref
2150          \let\ref\bbl@temp@ref
2151          \@safe@activestrue
2152          #3}%
2153       }%
2154     }{}%
2155   }

```

#### 10.4.2 varioref

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

```

\Ref 2156 \AtBeginDocument{%
2157   \ifpackageloaded{varioref}{%
2158     \bbl@redefine\@@vpageref#1[#2]#3{%
2159       \@safe@activestrue
2160       \org@@@vpageref{#1}[#2]{#3}%
2161       \@safe@activestrue}%

```

The same needs to happen for `\vrefpagemum`.

```

2162     \bbl@redefine\vrefpagemum#1#2{%

```

```

2163      \@safe@activetrue
2164      \org@vrefpagenum{#1}{#2}%
2165      \@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.

```

2166      \expandafter\def\csname Ref \endcsname#1{%
2167      \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
2168      }{}%
2169    }
2170 \fi

```

#### 10.4.3 `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.

```

2171 \AtEndOfPackage{%
2172   \AtBeginDocument{%
2173     \ifpackageloaded{hhline}%

```

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

```

2174     {\expandafter\ifx\csname normal@char:string\endcsname\relax
2175     \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.

```

2176       \makeatletter
2177       \def\@currname{hhline}\input{hhline.sty}\makeatother
2178     \fi}%
2179   }}}}

```

#### 10.4.4 `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.

```

2180 \AtBeginDocument{%
2181   \ifx\pdfstringdefDisableCommands\undefined\else
2182     \pdfstringdefDisableCommands{\languageshorthands{system}}%
2183   \fi}

```

#### 10.4.5 `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.

```

2184 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
2185   \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.

```

2186 \def\substitutefontfamily#1#2#3{%
2187   \lowercase{\immediate\openout15=#1#2.fd\relax}%
2188   \immediate\write15{%
2189     \string\ProvidesFile{#1#2.fd}%
2190     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
2191     \space generated font description file]^J
2192     \string\DeclareFontFamily{#1}{#2}{^^J
2193     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
2194     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
2195     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
2196     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
2197     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
2198     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
2199     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
2200     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
2201   }%
2202   \closeout15
2203 }

```

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

```

2204 \@onlypreamble\substitutefontfamily

```

## 10.5 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of  $\text{\TeX}$  and  $\text{\LaTeX}$  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`

```

2205 \bbl@trace{Encoding and fonts}
2206 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,}
2207 \let\org@TeX\TeX
2208 \let\org@LaTeX\LaTeX
2209 \let\ensureascii\@firstofone
2210 \AtBeginDocument{%
2211   \in@false
2212   \bbl@foreach\BabelNonASCII{% is there a non-ascii enc?
2213     \ifin@false
2214       \lowercase{\bbl@xin@{,#1enc.def,}{,\@filelist,}}%
2215       \fi}%
2216   \ifin@ % if a non-ascii has been loaded
2217     \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
2218     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
2219     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
2220     \def\bbl@tempb#1\@@{\uppercase{\bbl@tempc#1}ENC.DEF\empty\@@}%
2221     \def\bbl@tempc#1ENC.DEF#2\@@{%
2222       \ifx\@empty#2\else
2223         \bbl@ifunset{T#1}%
2224         {}%
2225         {\bbl@xin@{,#1,}{,\BabelNonASCII,}}%
2226       \ifin@

```

```

2227         \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
2228         \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
2229     \else
2230         \def\ensureascii#1{{\fontencoding{#1}\selectfont#1}}%
2231     \fi}%
2232 \fi}%
2233 \bbl@foreach\@filelist{\bbl@tempb#1\@@}% TODO - \@@ de mas??
2234 \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,}%
2235 \ifin@ \else
2236     \edef\ensureascii#1{%
2237         \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
2238 \fi
2239 \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.

```

2240 \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.

```

2241 \AtBeginDocument{%
2242     \@ifpackageloaded{fontspec}%
2243     {\xdef\latinencoding{%
2244         \ifx\UTFencname\@undefined
2245             EU\ifcase\bbl@engine\or2\or1\fi
2246         \else
2247             \UTFencname
2248         \fi}}%
2249     {\gdef\latinencoding{OT1}%
2250         \ifx\cf@encoding\bbl@t@one
2251             \xdef\latinencoding{\bbl@t@one}%
2252         \else
2253             \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}}%
2254     \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.

```

2255 \DeclareRobustCommand{\latintext}{%
2256     \fontencoding{\latinencoding}\selectfont
2257     \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.

```

2258 \ifx\@undefined\DeclareTextFontCommand
2259     \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
2260 \else
2261     \DeclareTextFontCommand{\textlatin}{\latintext}
2262 \fi

```

## 10.6 Basic bidi support

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

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

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

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

```
2263 \bbl@trace{Basic (internal) bidi support}
2264 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
2265 \def\bbl@rscripts{%
2266   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
2267   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
2268   Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
2269   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
2270   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
2271   Old South Arabian,}%
2272 \def\bbl@provide@dirs#1{%
2273   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
2274   \ifin@
2275     \global\bbl@csarg\chardef{wdir@#1}\@ne
2276     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
2277     \ifin@
2278       \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
2279     \fi
2280   \else
2281     \global\bbl@csarg\chardef{wdir@#1}\z@
2282   \fi}
2283 \def\bbl@switchdir{%
2284   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}}%
2285   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}}%
2286   \bbl@exp{\bbl@setdirs\bbl@cs{wdir@\languagename}}}%
2287 \def\bbl@setdirs#1{% TODO - math
2288   \ifcase\bbl@select@type % TODO - strictly, not the right test
2289     \bbl@bodydir{#1}%
2290     \bbl@paddir{#1}%
2291   \fi
2292   \bbl@textdir{#1}}
```



```

2293 \ifodd\bbl@engine % luatex=1
2294 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2295 \DisableBabelHook{babel-bidi}
2296 \chardef\bbl@thepardir\z@
2297 \def\bbl@getluadir#1{%
2298   \directlua{
2299     if tex.#1dir == 'TLT' then
2300       tex.sprint('0')
2301     elseif tex.#1dir == 'TRT' then
2302       tex.sprint('1')
2303     end}}
2304 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
2305   \ifcase#3\relax
2306     \ifcase\bbl@getluadir{#1}\relax\else
2307       #2 TLT\relax
2308     \fi
2309   \else
2310     \ifcase\bbl@getluadir{#1}\relax
2311       #2 TRT\relax
2312     \fi
2313   \fi}
2314 \def\bbl@textdir#1{%
2315   \bbl@setluadir{text}\textdir{#1}% TODO - ?\linedir
2316   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
2317 \def\bbl@pardir#1{\bbl@setluadir{par}\pardir{#1}%
2318   \chardef\bbl@thepardir#1\relax}
2319 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
2320 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
2321 \def\bbl@dirparastext{\pardir\the\textdir\relax}% %%%
2322 \else % pdftex=0, xetex=2
2323   \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2324   \DisableBabelHook{babel-bidi}
2325   \newcount\bbl@dirlevel
2326   \chardef\bbl@thetextdir\z@
2327   \chardef\bbl@thepardir\z@
2328   \def\bbl@textdir#1{%
2329     \ifcase#1\relax
2330       \chardef\bbl@thetextdir\z@
2331       \bbl@textdir@i\beginL\endL
2332     \else
2333       \chardef\bbl@thetextdir@ne
2334       \bbl@textdir@i\beginR\endR
2335     \fi}
2336   \def\bbl@textdir@i#1#2{%
2337     \ifhmode
2338       \ifnum\currentgrouplevel>\z@
2339         \ifnum\currentgrouplevel=\bbl@dirlevel
2340           \bbl@error{Multiple bidi settings inside a group}%
2341           {I'll insert a new group, but expect wrong results.}%
2342           \bgroup\aftergroup#2\aftergroup\egroup
2343         \else
2344           \ifcase\currentgrouptype\or % 0 bottom
2345             \aftergroup#2% 1 simple {}
2346           \or
2347             \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
2348           \or
2349             \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
2350           \or\or\or % vbox vtop align
2351           \or

```

```

2352         \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
2353         \or\or\or\or\or\or % output math disc insert vcent mathchoice
2354         \or
2355         \aftergroup#2% 14 \beginngroup
2356         \else
2357         \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
2358         \fi
2359         \fi
2360         \bbl@dirlevel\currentgrouplevel
2361         \fi
2362         #1%
2363         \fi}
2364 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
2365 \let\bbl@bodydir\@gobble
2366 \let\bbl@pagedir\@gobble
2367 \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).

```

2368 \def\bbl@xebidipar{%
2369   \let\bbl@xebidipar\relax
2370   \TeXeTstate\@ne
2371   \def\bbl@xeverypar{%
2372     \ifcase\bbl@thepardir
2373       \ifcase\bbl@thetextdir\else\beginR\fi
2374     \else
2375       {\setbox\z@\lastbox\beginR\box\z@}%
2376       \fi}%
2377   \let\bbl@severypar\everypar
2378   \newtoks\everypar
2379   \everypar=\bbl@severypar
2380   \bbl@severypar{\bbl@xeverypar\the\everypar}}
2381 \fi

```

A tool for weak L (mainly digits).

```

2382 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}

```

## 10.7 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`.

```

2383 \bbl@trace{Local Language Configuration}
2384 \ifx\loadlocalcfg\@undefined
2385   \@ifpackagewith{babel}{noconfigs}%
2386   {\let\loadlocalcfg\@gobble}%
2387   {\def\loadlocalcfg#1{%
2388     \InputIfFileExists{#1.cfg}%
2389     {\typeout{*****^J%
2390               * Local config file #1.cfg used^^J%
2391               *}}%
2392     \@empty}}
2393 \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:

```

2394 \ifx\@unexpandable@protect\@undefined
2395   \def\@unexpandable@protect{\noexpand\protect\noexpand}
2396   \long\def\protected@write#1#2#3{%
2397     \begingroup
2398       \let\thepage\relax
2399       #2%
2400       \let\protect\@unexpandable@protect
2401       \edef\reserved@a{\write#1{#3}}%
2402       \reserved@a
2403     \endgroup
2404     \if@nobreak\ifvmode\nobreak\fi\fi}
2405 \fi
2406 </core>
2407 <*kernel>

```

## 11 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.

```

2408 <<Make sure ProvidesFile is defined>>
2409 \ProvidesFile{switch.def}[\<date>] [\<version>] Babel switching mechanism]
2410 <<Load macros for plain if not LaTeX>>
2411 <<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.

```

2412 \def\bbl@version{\<version>}
2413 \def\bbl@date{\<date>}
2414 \def\adddialect#1#2{%
2415   \global\chardef#1#2\relax
2416   \bbl@usehooks{adddialect}{\#1}{\#2}}%
2417   \wlog{\string#1 = a dialect from \string\language#2}}

```

`\bbl@iflanguage` executes code only if the language `l@` exists. Otherwise raises an error. The argument of `\bbl@fixname` has to be a macro name, as it may get “fixed” if casing (lc/uc) is wrong. It’s 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.

```

2418 \def\bbl@fixname#1{%
2419   \begingroup
2420     \def\bbl@tempe{#1}%
2421     \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
2422     \bbl@tempd
2423     {\lowercase\expandafter{\bbl@tempd}%
2424      {\uppercase\expandafter{\bbl@tempd}%
2425       \@empty
2426       {\edef\bbl@tempd{\def\noexpand#1{#1}}%
2427        \uppercase\expandafter{\bbl@tempd}}}%
2428      {\edef\bbl@tempd{\def\noexpand#1{#1}}%
2429       \lowercase\expandafter{\bbl@tempd}}}%
2430     \@empty

```

```

2431 \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
2432 \bbl@tempd}
2433 \def\bbl@iflanguage#1{%
2434 \@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.

```

2435 \def\iflanguage#1{%
2436 \bbl@iflanguage{#1}{%
2437 \ifnum\csname l@#1\endcsname=\language
2438 \expandafter\@firstoftwo
2439 \else
2440 \expandafter\@secondoftwo
2441 \fi}}

```

## 11.1 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.

```

2442 \let\bbl@select@type\z@
2443 \edef\selectlanguage{%
2444 \noexpand\protect
2445 \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`.

```

2446 \ifx\@undefined\protect\let\protect\relax\fi

```

As  $\LaTeX$  2.09 writes to files *expanded* whereas  $\LaTeX$  2 $\epsilon$  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.

```

2447 \ifx\documentclass\@undefined
2448 \def\xstring{\string\string\string}
2449 \else
2450 \let\xstring\string
2451 \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 T<sub>E</sub>X'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.

```
2452 \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`

```
2453 \def\bbl@push@language{%
```

```
2454 \xdef\bbl@language@stack{\language+\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 `\language`. For this we first define a helper function.

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

```
2455 \def\bbl@pop@lang#1+#2-#3{%
```

```
2456 \edef\language{#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 T<sub>E</sub>X 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.

```
2457 \let\bbl@ifrestoring\@secondoftwo
```

```
2458 \def\bbl@pop@language{%
```

```
2459 \expandafter\bbl@pop@lang\bbl@language@stack-\bbl@language@stack
```

```
2460 \let\bbl@ifrestoring\@firstoftwo
```

```
2461 \expandafter\bbl@set@language\expandafter{\language}%
```

```
2462 \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.

```
2463 \expandafter\def\csname selectlanguage \endcsname#1{%
```

```
2464 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@\fi
```

```
2465 \bbl@push@language
```

```
2466 \aftergroup\bbl@pop@language
```

```
2467 \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.

```

2468 \def\BabelContentsFiles{toc,lof,lot}
2469 \def\bbl@set@language#1{%
2470   \edef\language{%
2471     \ifnum\escapechar=\expandafter`\string#1\@empty
2472     \else\string#1\@empty\fi}%
2473   \select@language{\language}%
2474   \expandafter\ifx\csname date\language\endcsname\relax\else
2475     \if@filesw
2476       \protected@write\@auxout{}\string\babel@aux{\language}{}%
2477       \bbl@usehooks{write}{}%
2478     \fi
2479   \fi}
2480 \def\select@language#1{%
2481   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
2482   \edef\language{#1}%
2483   \bbl@fixname\language
2484   \bbl@iflanguage\language{%
2485     \expandafter\ifx\csname date\language\endcsname\relax
2486       \bbl@error
2487       {Unknown language `#1'. Either you have\\%
2488        misspelled its name, it has not been installed,\\%
2489        or you requested it in a previous run. Fix its name,\\%
2490        install it or just rerun the file, respectively. In\\%
2491        some cases, you may need to remove the aux file}%
2492       {You may proceed, but expect wrong results}%
2493     \else
2494       \let\bbl@select@type\z@
2495       \expandafter\bbl@switch\expandafter{\language}%
2496     \fi}}
2497 \def\babel@aux#1#2{%
2498   \expandafter\ifx\csname date#1\endcsname\relax
2499     \expandafter\ifx\csname bbl@auxwarn#1\endcsname\relax
2500       \@namedef{bbl@auxwarn#1}{}%
2501       \bbl@warning
2502       {Unknown language `#1'. Very likely you\\%
2503        requested it in a previous run. Expect some\\%
2504        wrong results in this run, which should vanish\\%
2505        in the next one. Reported}%
2506     \fi
2507   \else
2508     \select@language{#1}%
2509     \bbl@foreach\BabelContentsFiles{%
2510       \@writefile{##1}{\babel@toc{#1}{#2}}}% % TODO - ok in plain?
2511   \fi}
2512 \def\babel@toc#1#2{%
2513   \select@language{#1}}

```

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

```

2514 \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 `\csname` primitive.

Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of `\selectlanguage`, and calling these macros.

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

```

2515 \newif\ifbbl@usedategroup
2516 \def\bbl@switch#1{%
2517   \originalTeX
2518   \expandafter\def\expandafter\originalTeX\expandafter{%
2519     \csname noextras#1\endcsname
2520     \let\originalTeX\empty
2521     \babel@beginsave}%
2522   \bbl@usehooks{afterreset}}}%
2523   \languageshorthands{none}%
2524   \ifcase\bbl@select@type
2525     \ifhmode
2526       \hskip\z@skip % trick to ignore spaces
2527       \csname captions#1\endcsname\relax
2528       \csname date#1\endcsname\relax
2529       \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
2530     \else
2531       \csname captions#1\endcsname\relax
2532       \csname date#1\endcsname\relax
2533     \fi
2534   \else\ifbbl@usedategroup
2535     \bbl@usedategroupfalse
2536     \ifhmode
2537       \hskip\z@skip % trick to ignore spaces
2538       \csname date#1\endcsname\relax
2539       \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
2540     \else
2541       \csname date#1\endcsname\relax
2542     \fi
2543   \fi\fi
2544   \bbl@usehooks{beforeextras}}}%
2545   \csname extras#1\endcsname\relax
2546   \bbl@usehooks{afterextras}}}%
2547   \ifcase\bbl@opt@hyphenmap\or
2548     \def\BabelLower##1##2{\lccode##1=##2\relax}%
2549     \ifnum\bbl@hymapsel>4\else
2550       \csname\language @bbl@hyphenmap\endcsname
2551     \fi
2552     \chardef\bbl@opt@hyphenmap\z@
2553   \else
2554     \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
2555       \csname\language @bbl@hyphenmap\endcsname
2556     \fi
2557   \fi
2558   \global\let\bbl@hymapsel@cclv
2559   \bbl@patterns{#1}%

```

```

2560 \babel@savevariable\lefthyphenmin
2561 \babel@savevariable\righthyphenmin
2562 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2563 \set@hyphenmins\tw@thr@\relax
2564 \else
2565 \expandafter\expandafter\expandafter\set@hyphenmins
2566 \csname #1hyphenmins\endcsname\relax
2567 \fi}

```

`otherlanguage` The `otherlanguage` environment can be used as an alternative to using the `\selectlanguage` declarative command. When you are typesetting a document which mixes left-to-right and right-to-left typesetting you have to use this environment in order to let things work as you expect them to. The `\ignorespaces` command is necessary to hide the environment when it is entered in horizontal mode.

```

2568 \long\def\otherlanguage#1{%
2569 \ifnum\bbl@hymapsel=\cclv\let\bbl@hymapsel\thr@\fi
2570 \csname selectlanguage\endcsname{#1}%
2571 \ignorespaces}

```

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

```

2572 \long\def\endotherlanguage{%
2573 \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`.

```

2574 \expandafter\def\csname otherlanguage*\endcsname#1{%
2575 \ifnum\bbl@hymapsel=\cclv\chardef\bbl@hymapsel4\relax\fi
2576 \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”.

```

2577 \expandafter\let\csname endotherlanguage*\endcsname\relax

```

`\foreignlanguage` The `\foreignlanguage` command is another substitute for the `\selectlanguage` command. This command takes two arguments, the first argument is the name of the language to use for typesetting the text specified in the second argument. Unlike `\selectlanguage` this command doesn’t switch *everything*, it only switches the hyphenation rules and the extra definitions for the language specified. It does this within a group and assumes the `\extras⟨lang⟩` command doesn’t make any `\global` changes. The coding is very similar to part of `\selectlanguage`. `\bbl@beforeforeign` is a trick to fix a bug in bidi texts. `\foreignlanguage` is supposed to be a ‘text’ command, and therefore it must emit a `\leavevmode`, but it does not, and therefore the indent is placed on the opposite margin. For backward compatibility, however, it is done only if a right-to-left script is requested; otherwise, it is no-op. (3.11) `\foreignlanguage*` is a temporary, experimental macro for a few lines with a different script direction, while preserving the paragraph format (thank the braces around `\par`, things like `\hangindent` are not reset). Do not use it in production, because its semantics and its syntax may change (and very likely will, or even it could be removed altogether). Currently it enters in `vmode` and then selects the language (which in turn sets the paragraph direction). (3.11) Also experimental are the hook `foreign` and `foreign*`. With them you can redefine `\BabelText` which by default does nothing. Its behavior is not well defined yet. So, use it in horizontal mode only if you do not want surprises.



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

```

2578 \providecommand\bb1@beforeforeign{
2579 \edef\foreignlanguage{%
2580   \noexpand\protect
2581   \expandafter\noexpand\csname foreignlanguage \endcsname}
2582 \expandafter\def\csname foreignlanguage \endcsname{%
2583   \@ifstar\bb1@foreign@s\bb1@foreign@x}
2584 \def\bb1@foreign@x#1#2{%
2585   \begingroup
2586     \let\BabelText\@firstofone
2587     \bb1@beforeforeign
2588     \foreign@language{#1}%
2589     \bb1@usehooks{foreign}{}%
2590     \BabelText{#2}% Now in horizontal mode!
2591   \endgroup}
2592 \def\bb1@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@par
2593   \begingroup
2594     {\par}%
2595     \let\BabelText\@firstofone
2596     \foreign@language{#1}%
2597     \bb1@usehooks{foreign*}{}%
2598     \bb1@dirparastext
2599     \BabelText{#2}% Still in vertical mode!
2600     {\par}%
2601   \endgroup}

```

`\foreign@language` This macro does the work for `\foreignlanguage` and the `otherlanguage*` environment. First we need to store the name of the language and check that it is a known language. Then it just calls `bb1@switch`.

```

2602 \def\foreign@language#1{%
2603   \edef\language#1%
2604   \bb1@fixname\language
2605   \bb1@iflanguage\language{%
2606     \expandafter\ifx\csname date\language\endcsname\relax
2607       \bb1@warning
2608       {Unknown language `#1'. Either you have\\%
2609        misspelled its name, it has not been installed,\\%
2610        or you requested it in a previous run. Fix its name,\\%
2611        install it or just rerun the file, respectively.\\%
2612        I'll proceed, but expect wrong results.\\%
2613        Reported}%
2614     \fi
2615     \let\bb1@select@type\@ne
2616     \expandafter\bb1@switch\expandafter{\language}}

```

`\bb1@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). `\bb1@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.

```

2617 \let\bb1@hyphlist\@empty
2618 \let\bb1@hyphenation@\relax

```

```

2619 \let\bbl@pttnlist\@empty
2620 \let\bbl@patterns@\relax
2621 \let\bbl@hymapsel=\cclv
2622 \def\bbl@patterns#1{%
2623   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
2624     \csname l@#1\endcsname
2625     \edef\bbl@tempa{#1}%
2626   \else
2627     \csname l@#1:\f@encoding\endcsname
2628     \edef\bbl@tempa{#1:\f@encoding}%
2629   \fi
2630   \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}}%
2631   \@ifundefined{bbl@hyphenation@}{#1}{% Can be \relax!
2632     \begingroup
2633       \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
2634       \ifin@else
2635         \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}}%
2636       \hyphenation{%
2637         \bbl@hyphenation@
2638         \@ifundefined{bbl@hyphenation@#1}%
2639         \@empty
2640         {\space\csname bbl@hyphenation@#1\endcsname}}%
2641       \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
2642     \fi
2643   \endgroup}}

```

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

```

2644 \def\hyphenrules#1{%
2645   \edef\bbl@tempf{#1}%
2646   \bbl@fixname\bbl@tempf
2647   \bbl@iflanguage\bbl@tempf{%
2648     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
2649     \languageshortands{none}%
2650     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
2651       \set@hyphenmins\tw@\thr@@\relax
2652     \else
2653       \expandafter\expandafter\expandafter\set@hyphenmins
2654       \csname\bbl@tempf hyphenmins\endcsname\relax
2655     \fi}}
2656 \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.

```

2657 \def\providehyphenmins#1#2{%
2658   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2659     \@namedef{#1hyphenmins}{#2}%
2660   \fi}

```

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

```

2661 \def\set@hyphenmins#1#2{%
2662   \lefthyphenmin#1\relax
2663   \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.

```

2664 \ifx\ProvidesFile\@undefined
2665   \def\ProvidesLanguage#1[#2 #3 #4]{%
2666     \wlog{Language: #1 #4 #3 <#2>}%
2667   }
2668 \else
2669   \def\ProvidesLanguage#1{%
2670     \begingroup
2671       \catcode`\ 10 %
2672       \@makeother\/%
2673       \@ifnextchar[%]
2674         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
2675   \def\@provideslanguage#1[#2]{%
2676     \wlog{Language: #1 #2}%
2677     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
2678     \endgroup}
2679 \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`

```

2680 \def\LdfInit{%
2681   \chardef\atcatcode=\catcode`\@
2682   \catcode`\@=11\relax
2683   \input babel.def\relax
2684   \catcode`\@=\atcatcode \let\atcatcode\relax
2685   \LdfInit}

```

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

```

2686 \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.

```

2687 \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’:

```

2688 \providecommand\setlocale{%
2689   \bbl@error
2690   {Not yet available}%
2691   {Find an armchair, sit down and wait}}
2692 \let\uselocale\setlocale
2693 \let\locale\setlocale
2694 \let\selectlocale\setlocale
2695 \let\textlocale\setlocale
2696 \let\textlanguage\setlocale
2697 \let\languagetext\setlocale

```

## 11.2 Errors

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

`\@noopterr` When the package was loaded without options not everything will work as expected. An error message is issued in that case.  
When the format knows about `\PackageError` it must be  $\text{\LaTeX 2}_{\epsilon}$ , so we can safely use its error handling interface. Otherwise we'll have to 'keep it simple'.

```

2698 \edef\bbl@nulllanguage{\string\language=0}
2699 \ifx\PackageError\@undefined
2700   \def\bbl@error#1#2{%
2701     \begingroup
2702       \newlinechar=`^^J
2703       \def\{^^J(babel) }%
2704       \errhelp{#2}\errmessage{\{#1}%
2705     \endgroup}
2706   \def\bbl@warning#1{%
2707     \begingroup
2708       \newlinechar=`^^J
2709       \def\{^^J(babel) }%
2710       \message{\{#1}%
2711     \endgroup}
2712   \def\bbl@info#1{%
2713     \begingroup
2714       \newlinechar=`^^J
2715       \def\{^^J}%
2716       \wlog{#1}%
2717     \endgroup}
2718 \else
2719   \def\bbl@error#1#2{%
2720     \begingroup
2721       \def\{\MessageBreak}%
2722       \PackageError{babel}{#1}{#2}%
2723     \endgroup}
2724   \def\bbl@warning#1{%
2725     \begingroup
2726       \def\{\MessageBreak}%
2727       \PackageWarning{babel}{#1}%
2728     \endgroup}
2729   \def\bbl@info#1{%
2730     \begingroup
2731       \def\{\MessageBreak}%
2732       \PackageInfo{babel}{#1}%
2733     \endgroup}
2734 \fi
2735 \@ifpackagewith{babel}{silent}
2736   {\let\bbl@info\@gobble
2737    \let\bbl@warning\@gobble}
2738   {}
2739 \def\bbl@nocaption{\protect\bbl@nocaption@i}
2740 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
2741   \global\@namedef{#2}{\textbf{?#1?}}%
2742   \@nameuse{#2}%
2743   \bbl@warning{%

```

```

2744 \backslashchar#2 not set. Please, define\\%
2745 it in the preamble with something like:\\%
2746 \string\renewcommand\backslashchar#2{..}\\%
2747 Reported}}
2748 \def\nolanerr#1{%
2749 \bbl@error
2750 {You haven't defined the language #1\space yet}%
2751 {Your command will be ignored, type <return> to proceed}}
2752 \def\nopatterns#1{%
2753 \bbl@warning
2754 {No hyphenation patterns were preloaded for\\%
2755 the language `#1' into the format.\\%
2756 Please, configure your TeX system to add them and\\%
2757 rebuild the format. Now I will use the patterns\\%
2758 preloaded for \bbl@nulllanguage\space instead}}
2759 \let\bbl@usehooks\gobbletwo
2760 </kernel>
2761 < *patterns>

```

## 12 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.

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  $\text{\LaTeX}$  the above scheme won't work. The reason is that  $\text{\LaTeX}$  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 `\org@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.

```

2762 <<Make sure ProvidesFile is defined>>
2763 \ProvidesFile{hyphen.cfg}[\<date>] \<version>] Babel hyphens]
2764 \xdef\bb1@format{\jobname}
2765 \ifx\AtBeginDocument\@undefined
2766   \def\@empty{}
2767   \let\orig@dump\dump
2768   \def\dump{%
2769     \ifx\@ztryfc\@undefined
2770     \else
2771       \toks0=\expandafter{\@preamblecmds}%
2772       \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
2773       \def\@begindocumenthook{}}%
2774     \fi
2775     \let\dump\orig@dump\let\orig@dump\@undefined\dump}
2776 \fi
2777 <<Define core switching macros>>

```

`\process@line` Each line in the file `language.dat` is processed by `\process@line` after it is read. The first thing this macro does is to check whether the line starts with `=`. When the first token of a line is an `=`, the macro `\process@synonym` is called; otherwise the macro `\process@language` will continue.

```

2778 \def\process@line#1#2 #3 #4 {%
2779   \ifx=#1%
2780     \process@synonym{#2}%
2781   \else
2782     \process@language{#1#2}{#3}{#4}%
2783   \fi
2784   \ignorespaces}

```

`\process@synonym` This macro takes care of the lines which start with an `=`. It needs an empty token register to begin with. `\bb1@languages` is also set to empty.

```

2785 \toks@{}
2786 \def\bb1@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.

```

2787 \def\process@synonym#1{%
2788   \ifnum\last@language=\m@ne
2789     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
2790   \else
2791     \expandafter\chardef\csname l@#1\endcsname\last@language
2792     \wlog{\string\l@#1=\string\language\the\last@language}%
2793     \expandafter\let\csname #1hyphenmins\endcsname\expandafter\endcsname
2794     \csname\language\endcsname hyphenmins\endcsname
2795     \let\bb1@elt\relax
2796     \edef\bb1@languages{\bb1@languages\bb1@elt{#1}{\the\last@language}}}%
2797   \fi}

```

`\process@language` The macro `\process@language` is used to process a non-empty line from the ‘configuration file’. It has three arguments, each delimited by white space. The first argument is the ‘name’ of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions. The first thing to do is call `\addlanguage` to allocate a pattern register and to make that register ‘active’. Then the pattern file is read.

For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file `language.dat` by adding for instance ‘:T1’ to the name of the language. The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

Pattern files may contain assignments to `\lefthyphenmin` and `\righthyphenmin`. T<sub>E</sub>X does not keep track of these assignments. Therefore we try to detect such assignments and store them in the `\<lang>hyphenmins` macro. When no assignments were made we provide a default setting.

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

Then we globally store the settings of `\lefthyphenmin` and `\righthyphenmin` and close the group.

When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

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

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

```

2798 \def\process@language#1#2#3{%
2799   \expandafter\addlanguage\csname l@#1\endcsname
2800   \expandafter\language\csname l@#1\endcsname
2801   \edef\language#1#2#3{%
2802     \bbl@hook@everylanguage{#1}%
2803     \bbl@get@enc#1::\@@@
2804     \begingroup
2805       \lefthyphenmin\m@ne
2806       \bbl@hook@loadpatterns{#2}%
2807       \ifnum\lefthyphenmin=\m@ne
2808       \else
2809         \expandafter\xdef\csname #1hyphenmins\endcsname{%
2810           \the\lefthyphenmin\the\righthyphenmin}%
2811       \fi
2812     \endgroup
2813   \def\bbl@tempa{#3}%
2814   \ifx\bbl@tempa\@empty\else
2815     \bbl@hook@loadexceptions{#3}%
2816   \fi
2817   \let\bbl@elt\relax
2818   \edef\bbl@languages{%
2819     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
2820   \ifnum\the\language=\z@
2821     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2822       \set@hyphenmins\tw@\thr@@\relax
2823     \else
2824       \expandafter\expandafter\expandafter\set@hyphenmins
2825       \csname #1hyphenmins\endcsname
2826     \fi
2827     \the\toks@
2828     \toks@{}%
2829   \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.

```
2830 \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.

```
2831 \def\bbl@hook@everylanguage#1{}
2832 \def\bbl@hook@loadpatterns#1{\input #1\relax}
2833 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
2834 \let\bbl@hook@loadkernel\bbl@hook@loadpatterns
2835 \begingroup
2836   \def\AddBabelHook#1#2{%
2837     \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
2838       \def\next{\toks1}%
2839     \else
2840       \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
2841     \fi
2842     \next}
2843 \ifx\directlua\undefined
2844   \ifx\XeTeXinputencoding\undefined\else
2845     \input xebabel.def
2846   \fi
2847 \else
2848   \input luababel.def
2849 \fi
2850 \openin1 = babel-\bbl@format.cfg
2851 \ifeof1
2852 \else
2853   \input babel-\bbl@format.cfg\relax
2854 \fi
2855 \closein1
2856 \endgroup
2857 \bbl@hook@loadkernel{switch.def}
```

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

```
2858 \openin1 = language.dat
```

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

```
2859 \def\language{english}%
2860 \ifeof1
2861   \message{I couldn't find the file language.dat,\space
2862     I will try the file hyphen.tex}
2863   \input hyphen.tex\relax
2864   \chardef\l@english\z@
2865 \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  $-1$ .

```
2866   \last@language\m@ne
```

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

```
2867   \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.

```
2868 \endlinechar\m@ne
2869 \read1 to \bbl@line
2870 \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.

```
2871 \if T\ifeof1F\fi T\relax
2872 \ifx\bbl@line\@empty\else
2873 \edef\bbl@line{\bbl@line\space\space\space}%
2874 \expandafter\process@line\bbl@line\relax
2875 \fi
2876 \repeat
```

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

```
2877 \begingroup
2878 \def\bbl@elt#1#2#3#4{%
2879 \global\language=#2\relax
2880 \gdef\language#1}%
2881 \def\bbl@elt##1##2##3##4{}}%
2882 \bbl@languages
2883 \endgroup
2884 \fi
```

and close the configuration file.

```
2885 \closein1
```

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

```
2886 \if/\the\toks@/\else
2887 \errhelp{language.dat loads no language, only synonyms}
2888 \errmessage{0rphan language synonym}
2889 \fi
```

Also remove some macros from memory and raise an error if \toks@ is not empty. Finally load switch.def, but the latter is not required and the line inputting it may be commented out.

```
2890 \let\bbl@line\@undefined
2891 \let\process@line\@undefined
2892 \let\process@synonym\@undefined
2893 \let\process@language\@undefined
2894 \let\bbl@get@enc\@undefined
2895 \let\bbl@hyph@enc\@undefined
2896 \let\bbl@tempa\@undefined
2897 \let\bbl@hook@loadkernel\@undefined
2898 \let\bbl@hook@everylanguage\@undefined
2899 \let\bbl@hook@loadpatterns\@undefined
2900 \let\bbl@hook@loadexceptions\@undefined
2901 \</patterns>
```

Here the code for `initTeX` ends.

## 13 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].

```

2902 <<{*More package options}>> ≡
2903 \ifodd\bb1@engine
2904   \DeclareOption{bidi=basic-r}%
2905     {\ExecuteOptions{bidi=basic}}
2906   \DeclareOption{bidi=basic}%
2907     {\let\bb1@beforeforeign\leavevmode
2908       \newattribute\bb1@attr@dir
2909       \bb1@exp{\output{\bodydir\pagedir\the\output}}}%
2910     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}}
2911 \else
2912   \DeclareOption{bidi=basic-r}%
2913     {\ExecuteOptions{bidi=basic}}
2914   \DeclareOption{bidi=basic}%
2915     {\bb1@error
2916       {The bidi method 'basic' is available only in\\%
2917         luatex. I'll continue with 'bidi=default', so\\%
2918         expect wrong results}%
2919       {See the manual for further details.}%
2920       \let\bb1@beforeforeign\leavevmode
2921       \AtEndOfPackage{%
2922         \EnableBabelHook{babel-bidi}%
2923         \bb1@xebidipar}}
2924 \fi
2925 \DeclareOption{bidi=default}%
2926   {\let\bb1@beforeforeign\leavevmode
2927     \ifodd\bb1@engine
2928       \newattribute\bb1@attr@dir
2929       \bb1@exp{\output{\bodydir\pagedir\the\output}}}%
2930   \fi
2931   \AtEndOfPackage{%
2932     \EnableBabelHook{babel-bidi}%
2933     \ifodd\bb1@engine\else
2934       \bb1@xebidipar
2935     \fi}}
2936 <</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.

```

2937 <<{*Font selection}>> ≡
2938 \bb1@trace{Font handling with fontspec}
2939 \@onlypreamble\babelfont
2940 \newcommand\babelfont[2][]{% 1=langs/scripts 2=fam
2941   \edef\bb1@tempa{#1}%
2942   \def\bb1@tempb{#2}%
2943   \ifx\fontspec@undefined
2944     \usepackage{fontspec}%
2945   \fi
2946   \EnableBabelHook{babel-fontspec}%
2947   \bb1@bb1font}
2948 \newcommand\bb1@bb1font[2][]{% 1=features 2=fontname
2949   \bb1@ifunset{\bb1@tempb family}{\bb1@providefam{\bb1@tempb}}}%
2950   \bb1@ifunset{\bb1@lsys\languagename}{\bb1@provide@lsys{\languagename}}}%
2951   \expandafter\bb1@ifblank\expandafter{\bb1@tempa}%

```

```

2952 {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
2953 \bbl@exp{%
2954 \let<\bbl@bbl@tempb dflt@\language>\<\bbl@bbl@tempb dflt@>%
2955 \\\bbl@font@set<\bbl@bbl@tempb dflt@\language>%
2956 \<\bbl@tempb default>\<\bbl@tempb family>}}%
2957 {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
2958 \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:

```

2959 \def\bbl@providfam#1{%
2960 \bbl@exp{%
2961 \\\newcommand\<#1default>{}% Just define it
2962 \\\bbl@add@list\\bbl@font@fams{#1}%
2963 \\\DeclareRobustCommand\<#1family>%
2964 \\\not@math@alphabet\<#1family>\relax
2965 \\\fontfamily\<#1default>\\selectfont}%
2966 \\\DeclareTextFontCommand{\<text#1>}{\<#1family>}}

```

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

```

2967 \def\bbl@switchfont{%
2968 \bbl@ifunset{\bbl@lsys@\language}{\bbl@provide@lsys{\language}}}%
2969 \bbl@exp{% eg Arabic -> arabic
2970 \lowercase{\edef\\bbl@tempa{\bbl@cs{sname@\language}}}%
2971 \bbl@foreach\bbl@font@fams{%
2972 \bbl@ifunset{\bbl@##1dflt@\language}% (1) language?
2973 {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}% (2) from script?
2974 {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
2975 {}% 123=F - nothing!
2976 {\bbl@exp{% 3=T - from generic
2977 \global\let<\bbl@##1dflt@\language>%
2978 \<\bbl@##1dflt@>}}}%
2979 {\bbl@exp{% 2=T - from script
2980 \global\let<\bbl@##1dflt@\language>%
2981 \<\bbl@##1dflt@*\bbl@tempa>}}}%
2982 {}% 1=T - language, already defined
2983 \def\bbl@tempa{%
2984 \bbl@warning{The current font is not a standard family:\\%
2985 \fontname\font\\%
2986 Script and Language are not applied. Consider defining a\\%
2987 new family with \string\babelfont. Reported}}%
2988 \bbl@foreach\bbl@font@fams{% don't gather with prev for
2989 \bbl@ifunset{\bbl@##1dflt@\language}%
2990 {\bbl@cs{famrst@##1}%
2991 \global\bbl@csarg\let{famrst@##1}\relax}%
2992 {\bbl@exp{% order is relevant
2993 \\\bbl@add\\originalTeX{%
2994 \\\bbl@font@rst{\bbl@cs{##1dflt@\language}}%
2995 \<##1default>\<##1family>{##1}}%
2996 \\\bbl@font@set<\bbl@##1dflt@\language>% the main part!
2997 \<##1default>\<##1family>}}}%
2998 \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. We must deactivate temporarily \bbl@mapselect because \selectfont is called internally when a font is defined.

```

2999 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
3000 \bbl@xin@{<>}{#1}%

```

```

3001 \ifin@
3002 \bbl@exp{\bbl@fontspec@set\#1\expandafter\@gobbletwo#1}%
3003 \fi
3004 \bbl@exp{%
3005   \def\#2{#1}%          eg, \rmdefault{\bbl@rmdflt@lang}
3006   \bbl@ifsamestring{#2}{\f@family}{\#3\let\bbl@tempa\relax}{}}%
3007 \def\bbl@fontspec@set#1#2#3{% eg \bbl@rmdflt@lang fnt-opt fnt-nme
3008   \let\bbl@tempe\bbl@mapselect
3009   \let\bbl@mapselect\relax
3010   \bbl@exp{\<fontspec_set_family:Nnn>\#1%
3011     {\bbl@cs{lsys@\languagename},#2}}{#3}%
3012   \let\bbl@mapselect\bbl@tempe
3013   \bbl@toggleglobal#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.

```

3014 \def\bbl@font@rst#1#2#3#4{%
3015   \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.

```

3016 \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 :-).

```

3017 \newcommand\babelFSstore[2][{}]{%
3018   \bbl@ifblank{#1}%
3019   {\bbl@csarg\def{sname@#2}{Latin}}%
3020   {\bbl@csarg\def{sname@#2}{#1}}%
3021   \bbl@provide@dirs{#2}%
3022   \bbl@csarg\ifnum{wdir@#2}>\z@
3023     \let\bbl@beforeforeign\leavevmode
3024     \EnableBabelHook{babel-bidi}%
3025   \fi
3026   \bbl@foreach{#2}{%
3027     \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
3028     \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
3029     \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}%
3030 \def\bbl@FSstore#1#2#3#4{%
3031   \bbl@csarg\edef{#2default#1}{#3}%
3032   \expandafter\addto\csname extras#1\endcsname{%
3033     \let#4#3%
3034     \ifx#3\f@family
3035       \edef#3{\csname bbl@#2default#1\endcsname}%
3036       \fontfamily{#3}\selectfont
3037     \else
3038       \edef#3{\csname bbl@#2default#1\endcsname}%
3039     \fi}%
3040   \expandafter\addto\csname noextras#1\endcsname{%
3041     \ifx#3\f@family
3042       \fontfamily{#4}\selectfont
3043     \fi
3044     \let#3#4}}
3045 \let\bbl@langfeatures\@empty
3046 \def\babelFSfeatures{% make sure \fontspec is redefined once
3047   \let\bbl@ori@fontspec\fontspec
3048   \renewcommand\fontspec[1][{}]{%
3049     \bbl@ori@fontspec[\bbl@langfeatures##1]}

```

```

3050 \let\babelFSfeatures\bb1@FSfeatures
3051 \babelFSfeatures}
3052 \def\bb1@FSfeatures#1#2{%
3053 \expandafter\addto\csname extras#1\endcsname{%
3054 \babel@save\bb1@langfeatures
3055 \edef\bb1@langfeatures{#2,}}
3056 <</Font selection>>

```

## 14 Hooks for XeTeX and LuaTeX

### 14.1 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 Lua<sub>T</sub><sub>E</sub>X also resets the catcodes.

```

3057 <<(*Restore Unicode catcodes before loading patterns)>> ≡
3058 \begingroup
3059 % Reset chars "80-"C0 to category "other", no case mapping:
3060 \catcode`\@=11 \count@=128
3061 \loop\ifnum\count@<192
3062 \global\uccode\count@=0 \global\lccode\count@=0
3063 \global\catcode\count@=12 \global\sffcode\count@=1000
3064 \advance\count@ by 1 \repeat
3065 % Other:
3066 \def\0 ##1 {%
3067 \global\uccode"##1=0 \global\lccode"##1=0
3068 \global\catcode"##1=12 \global\sffcode"##1=1000 }%
3069 % Letter:
3070 \def\L ##1 ##2 ##3 {\global\catcode"##1=11
3071 \global\uccode"##1="##2
3072 \global\lccode"##1="##3
3073 % Uppercase letters have sffcode=999:
3074 \ifnum"##1="##3 \else \global\sffcode"##1=999 \fi }%
3075 % Letter without case mappings:
3076 \def\l ##1 {\L ##1 ##1 ##1 }%
3077 \l 00AA
3078 \L 00B5 039C 00B5
3079 \l 00BA
3080 \O 00D7
3081 \l 00DF
3082 \O 00F7
3083 \L 00FF 0178 00FF
3084 \endgroup
3085 \input #1\relax
3086 <</Restore Unicode catcodes before loading patterns>>

```

Some more common code.

```

3087 <<(*Footnote changes)>> ≡
3088 \bb1@trace{Bidi footnotes}
3089 \ifx\bb1@beforeforeign\leavevmode
3090 \def\bb1@footnote#1#2#3{%
3091 \@ifnextchar[%
3092 {\bb1@footnote@o{#1}{#2}{#3}}%
3093 {\bb1@footnote@x{#1}{#2}{#3}}}

```

```

3094 \def\bbl@footnote@x#1#2#3#4{%
3095   \bgroup
3096   \select@language@x{\bbl@main@language}%
3097   \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
3098   \egroup}
3099 \def\bbl@footnote@o#1#2#3[#4]#5{%
3100   \bgroup
3101   \select@language@x{\bbl@main@language}%
3102   \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
3103   \egroup}
3104 \def\bbl@footnotetext#1#2#3{%
3105   \@ifnextchar[%
3106     {\bbl@footnotetext@o{#1}{#2}{#3}}%
3107     {\bbl@footnotetext@x{#1}{#2}{#3}}}
3108 \def\bbl@footnotetext@x#1#2#3#4{%
3109   \bgroup
3110   \select@language@x{\bbl@main@language}%
3111   \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
3112   \egroup}
3113 \def\bbl@footnotetext@o#1#2#3[#4]#5{%
3114   \bgroup
3115   \select@language@x{\bbl@main@language}%
3116   \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
3117   \egroup}
3118 \def\BabelFootnote#1#2#3#4{%
3119   \ifx\bbl@fn@footnote\@undefined
3120     \let\bbl@fn@footnote\footnote
3121   \fi
3122   \ifx\bbl@fn@footnotetext\@undefined
3123     \let\bbl@fn@footnotetext\footnotetext
3124   \fi
3125   \bbl@ifblank{#2}%
3126   {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
3127    \@namedef{\bbl@stripslash#1text}%
3128    {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
3129   {\def#1{\bbl@exp{\bbl@footnote{\bbl@foreignlanguage{#2}}}{#3}{#4}}%
3130    \@namedef{\bbl@stripslash#1text}%
3131    {\bbl@exp{\bbl@footnotetext{\bbl@foreignlanguage{#2}}}{#3}{#4}}}%
3132 \fi
3133 <</Footnote changes>>

```

Now, the code.

```

3134 < *xetex >
3135 \def\BabelStringsDefault{unicode}
3136 \let\xebbl@stop\relax
3137 \AddBabelHook{xetex}{encodedcommands}{%
3138   \def\bbl@tempa{#1}%
3139   \ifx\bbl@tempa\@empty
3140     \XeTeXinputencoding"bytes"%
3141   \else
3142     \XeTeXinputencoding"#1"%
3143   \fi
3144   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
3145 \AddBabelHook{xetex}{stopcommands}{%
3146   \xebbl@stop
3147   \let\xebbl@stop\relax}
3148 \AddBabelHook{xetex}{loadkernel}{%
3149 <<Restore Unicode catcodes before loading patterns>>}
3150 \ifx\DisableBabelHook\@undefined\endinput\fi

```

```

3151 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
3152 \DisableBabelHook{babel-fontspec}
3153 <<Font selection>>
3154 \input txtbabel.def
3155 </xetex>

```

## 14.2 Layout

*In progress.*

Unfortunately, proper support for xetex requires patching somehow lots of macros and packages (and some issues related to \specials remain, like color and hyperlinks). At least at this stage, babel will not do it and therefore a package like bidi (by Vafa Khalighi) would be necessary to overcome the limitations of xetex. Any help in making babel and bidi collaborate will be welcome, although the underlying concepts in both packages seem very different. Note also elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titleps, and geometry.

\bbl@startskip and \bbl@endskip are available to package authors. Thanks to the T<sub>E</sub>X expansion mechanism the following constructs are valid: \adim\bbl@startskip, \advance\bbl@startskip\adim, \bbl@startskip\adim.

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

```

3156 <*texxet>
3157 \bbl@trace{Redefinitions for bidi layout}
3158 \def\bbl@sspre@caption{%
3159   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
3160 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
3161 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
3162 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
3163 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
3164   \def\@hangfrom#1{%
3165     \setbox\@tempboxa\hbox{#1}%
3166     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
3167     \noindent\box\@tempboxa}
3168 \def\raggedright{%
3169   \let\@centercr
3170   \bbl@startskip\z@skip
3171   \@rightskip\@flushglue
3172   \bbl@endskip\@rightskip
3173   \parindent\z@
3174   \parfillskip\bbl@startskip}
3175 \def\raggedleft{%
3176   \let\@centercr
3177   \bbl@startskip\@flushglue
3178   \bbl@endskip\z@skip
3179   \parindent\z@
3180   \parfillskip\bbl@endskip}
3181 \fi
3182 \IfBabelLayout{lists}
3183   {\def\list#1#2{%
3184     \ifnum \@listdepth >5\relax
3185       \@toodeep
3186     \else
3187       \global\advance\@listdepth\@ne
3188     \fi
3189     \rightmargin\z@
3190     \listparindent\z@
3191     \itemindent\z@

```

```

3192 \csname @list\romannumeral\the\@listdepth\endcsname
3193 \def\@itemlabel{#1}%
3194 \let\makelabel\@mklab
3195 \@nmbrlistfalse
3196 #2\relax
3197 \@trivlist
3198 \parskip\parsep
3199 \parindent\listparindent
3200 \advance\linewidth-\rightmargin
3201 \advance\linewidth-\leftmargin
3202 \advance\@totalleftmargin
3203 \ifcase\bb1\the\parindent\leftmargin\else\rightmargin\fi
3204 \parshape\@ne\@totalleftmargin\linewidth
3205 \ignorespaces}%
3206 \ifcase\bb1\@engine
3207 \def\labelenumii{}\theenumii}%
3208 \def\p@enumiii{\p@enumii}\theenumii}%
3209 \fi
3210 \def\@verbatim{%
3211 \trivlist \item\relax
3212 \if@minipage\else\vskip\parskip\fi
3213 \bb1\startskip\textwidth
3214 \advance\bb1\startskip-\linewidth
3215 \bb1\endskip\z@skip
3216 \parindent\z@
3217 \parfillskip\@flushglue
3218 \parskip\z@skip
3219 \@@par
3220 \language\@nohyphenation
3221 \@tempwafalse
3222 \def\par{%
3223 \if@tempwa
3224 \leavevmode\null
3225 \@@par\penalty\interlinepenalty
3226 \else
3227 \@tempwattrue
3228 \ifhmode\@@par\penalty\interlinepenalty\fi
3229 \fi}%
3230 \let\do\@makeother \dospecials
3231 \obeylines \verbatim@font \@noligs
3232 \everypar\expandafter{\the\everypar\unpenalty}}
3233 {}
3234 \IfBabelLayout{contents}
3235 {\def\@dottedtocline#1#2#3#4#5{%
3236 \ifnum#1>\c@tocdepth\else
3237 \vskip \z@ \@plus.2\p@
3238 {\bb1\startskip#2\relax
3239 \bb1\endskip\@tocrmarg
3240 \parfillskip-\bb1\endskip
3241 \parindent#2\relax
3242 \@afterindenttrue
3243 \interlinepenalty\@M
3244 \leavevmode
3245 \@tempdima#3\relax
3246 \advance\bb1\startskip\@tempdima
3247 \null\nobreak\hskip-\bb1\startskip
3248 {#4}\nobreak
3249 \leaders\hbox{%
3250 $ \m@th\mkern\@dotsep mu\hbox{.}\mkern\@dotsep mu$}%

```



```

3251         \hfill\nobreak
3252         \hb@xt@{\pnumwidth{\hfil\normalfont\normalcolor#5}}%
3253         \par}%
3254     \fi}}
3255 {}
3256 \IfBabelLayout{columns}
3257 {\def\@outputdblcol{%
3258     \if@firstcolumn
3259         \global\@firstcolumnfalse
3260         \global\setbox\@leftcolumn\copy\@outputbox
3261         \splitmaxdepth\maxdimen
3262         \vbadness\maxdimen
3263         \setbox\@outputbox\vbox{\unvbox\@outputbox\unskip}%
3264         \setbox\@outputbox\vsplit\@outputbox to\maxdimen
3265         \toks@{\expandafter{\topmark}}%
3266         \xdef\@firstcoltopmark{\the\toks@}%
3267         \toks@\expandafter{\splitfirstmark}%
3268         \xdef\@firstcolfirstmark{\the\toks@}%
3269         \ifx\@firstcolfirstmark\@empty
3270             \global\let\@setmarks\relax
3271         \else
3272             \gdef\@setmarks{%
3273                 \let\firstmark\@firstcolfirstmark
3274                 \let\topmark\@firstcoltopmark}%
3275         \fi
3276     \else
3277         \global\@firstcolumntrue
3278         \setbox\@outputbox\vbox{%
3279             \hb@xt@\textwidth{%
3280                 \hskip\columnwidth
3281                 \hfil
3282                 {\normalcolor\vrule \@width\columnseprule}%
3283                 \hfil
3284                 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
3285                 \hskip-\textwidth
3286                 \hb@xt@\columnwidth{\box\@outputbox \hss}%
3287                 \hskip\columnsep
3288                 \hskip\columnwidth}}%
3289         \@combinedblfloats
3290         \@setmarks
3291         \@outputpage
3292         \beginingroup
3293             \@dblfloatplacement
3294             \@startdblcolumn
3295             \@whilesw\if@fcolmade \fi{\@outputpage
3296                 \@startdblcolumn}%
3297         \endgroup
3298     \fi}}%
3299 {}
3300 <<Footnote changes>>
3301 \IfBabelLayout{footnotes}%
3302 {\BabelFootnote\footnote\language\language{}{}}%
3303 \BabelFootnote\localfootnote\language\language{}{}}%
3304 \BabelFootnote\mainfootnote{}{}{}}
3305 {}

```

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.

```

3306 \IfBabelLayout{counters}%

```

```

3307 {\let\bbl@latinarabic=\@arabic
3308 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
3309 \let\bbl@asciroman=\@roman
3310 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
3311 \let\bbl@asciiRoman=\@Roman
3312 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
3313 </texxet>

```

### 14.3 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 have been preloaded into the format. This is not optimal, but it shouldn't happen very often – with luatex patterns are best loaded when the document is typeset, and the "0th" language is preloaded just for backwards compatibility.

As of 1.1b, `lua(e)tex` is taken into account. Formerly, loading of patterns on the fly didn't work in this format, but with the new loader it does. Unfortunately, the format is not based on `babel`, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format `language.dat` is used (under the principle of a single source), instead of `language.def`.

Of course, there is room for improvements, like tools to read and reassign languages, which would require modifying the language list, and better error handling.

We need catcode tables, but no format (targeted by `babel`) provide a command to allocate them (although there are packages like `ctablestack`). 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.

```

3314 <*luatex>
3315 \ifx\AddBabelHook\@undefined
3316 \bbl@trace{Read language.dat}
3317 \begingroup
3318 \toks@{}
3319 \count@ \z@ % 0=start, 1=0th, 2=normal
3320 \def\bbl@process@line#1#2 #3 #4 {%
3321   \ifx=#1%
3322     \bbl@process@synonym{#2}%
3323   \else
3324     \bbl@process@language{#1#2}{#3}{#4}%
3325   \fi
3326   \ignorespaces}
3327 \def\bbl@manylang{%
3328   \ifnum\bbl@last>\@ne

```

```

3329     \bbl@info{Non-standard hyphenation setup}%
3330     \fi
3331     \let\bbl@manylang\relax}
3332 \def\bbl@process@language#1#2#3{%
3333     \ifcase\count@
3334         \@ifundefined{zth@#1}{\count@=\tw@}{\count@=\@ne}%
3335     \or
3336         \count@=\tw@
3337     \fi
3338     \ifnum\count@=\tw@
3339         \expandafter\addlanguage\csname l@#1\endcsname
3340         \language\allocationnumber
3341         \chardef\bbl@last\allocationnumber
3342         \bbl@manylang
3343         \let\bbl@elt\relax
3344         \xdef\bbl@languages{%
3345             \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
3346     \fi
3347     \the\toks@
3348     \toks@{}}
3349 \def\bbl@process@synonym@aux#1#2{%
3350     \global\expandafter\chardef\csname l@#1\endcsname#2\relax
3351     \let\bbl@elt\relax
3352     \xdef\bbl@languages{%
3353         \bbl@languages\bbl@elt{#1}{#2}{}}}%
3354 \def\bbl@process@synonym#1{%
3355     \ifcase\count@
3356         \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
3357     \or
3358         \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
3359     \else
3360         \bbl@process@synonym@aux{#1}{\the\bbl@last}%
3361     \fi}
3362 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
3363     \chardef\l@english\z@
3364     \chardef\l@USenglish\z@
3365     \chardef\bbl@last\z@
3366     \global\@namedef{\bbl@hyphendata@0}{\hyphen.tex}{}
3367     \gdef\bbl@languages{%
3368         \bbl@elt{english}{0}{\hyphen.tex}{}%
3369         \bbl@elt{USenglish}{0}{}}
3370 \else
3371     \global\let\bbl@languages@format\bbl@languages
3372     \def\bbl@elt#1#2#3#4{% Remove all except language 0
3373         \ifnum#2>\z@\else
3374             \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
3375         \fi}%
3376     \xdef\bbl@languages{\bbl@languages}%
3377     \fi
3378     \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
3379     \bbl@languages
3380     \openin1=language.dat
3381     \ifeof1
3382         \bbl@warning{I couldn't find language.dat. No additional\\%
3383             patterns loaded. Reported}%
3384     \else
3385         \loop
3386             \endlinechar\m@ne
3387             \read1 to \bbl@line

```

```

3388 \endlinechar\^^M
3389 \if T\ifeof1F\fi T\relax
3390 \ifx\bbbl@line\empty\else
3391 \edef\bbbl@line{\bbbl@line\space\space\space}%
3392 \expandafter\bbbl@process@line\bbbl@line\relax
3393 \fi
3394 \repeat
3395 \fi
3396 \endgroup
3397 \bbbl@trace{Macros for reading patterns files}
3398 \def\bbbl@get@enc#1:#2:#3\@@{\def\bbbl@hyph@enc{#2}}
3399 \ifx\babelcatcodetablenum\undefined
3400 \def\babelcatcodetablenum{5211}
3401 \fi
3402 \def\bbbl@luapatterns#1#2{%
3403 \bbbl@get@enc#1::\@@@
3404 \setbox\z@\hbox\bgroup
3405 \begingroup
3406 \ifx\catcodetable\undefined
3407 \let\savecatcodetable\luatexsavecatcodetable
3408 \let\initcatcodetable\luatexinitcatcodetable
3409 \let\catcodetable\luatexcatcodetable
3410 \fi
3411 \savecatcodetable\babelcatcodetablenum\relax
3412 \initcatcodetable\numexpr\babelcatcodetablenum+1\relax
3413 \catcodetable\numexpr\babelcatcodetablenum+1\relax
3414 \catcode`\_#6 \catcode`\$#3 \catcode`\&#4 \catcode`\^#7
3415 \catcode`\_#8 \catcode`\{#1 \catcode`\}=2 \catcode`\~#13
3416 \catcode`\@#11 \catcode`\^^I#10 \catcode`\^^J#12
3417 \catcode`\<#12 \catcode`\>#12 \catcode`\*#12 \catcode`\.=12
3418 \catcode`\-#12 \catcode`\/#12 \catcode`\[#12 \catcode`\]=12
3419 \catcode`\`#12 \catcode`\'#12 \catcode`\"#12
3420 \input #1\relax
3421 \catcodetable\babelcatcodetablenum\relax
3422 \endgroup
3423 \def\bbbl@tempa{#2}%
3424 \ifx\bbbl@tempa\empty\else
3425 \input #2\relax
3426 \fi
3427 \egroup}%
3428 \def\bbbl@patterns@lua#1{%
3429 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
3430 \csname l@#1\endcsname
3431 \edef\bbbl@tempa{#1}%
3432 \else
3433 \csname l@#1:\f@encoding\endcsname
3434 \edef\bbbl@tempa{#1:\f@encoding}%
3435 \fi\relax
3436 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
3437 \@ifundefined{bbbl@hyphendata@the\language}%
3438 {\def\bbbl@elt##1##2##3##4{%
3439 \ifnum##2=\csname l@bbbl@tempa\endcsname % #2=spanish, dutch:OT1...
3440 \def\bbbl@tempb{##3}%
3441 \ifx\bbbl@tempb\empty\else % if not a synonymous
3442 \def\bbbl@tempc{{##3}{##4}}%
3443 \fi
3444 \bbbl@csarg\xdef{hyphendata@##2}{\bbbl@tempc}%
3445 \fi}%
3446 \bbbl@languages

```

```

3447 \ifundefined{bbl@hyphendata@the\language}%
3448 {\bbl@info{No hyphenation patterns were set for\%
3449 language '\bbl@tempa'. Reported}}%
3450 {\expandafter\expandafter\expandafter\bbl@luapatterns
3451 \csname bbl@hyphendata@the\language\endcsname}}}}
3452 \endinput\fi
3453 \beginingroup
3454 \catcode`\%=12
3455 \catcode`\'=12
3456 \catcode`\%=12
3457 \catcode`\:=12
3458 \directlua{
3459 Babel = Babel or {}
3460 function Babel.bytes(line)
3461   return line:gsub("(.)",
3462     function (chr) return unicode.utf8.char(string.byte(chr)) end)
3463 end
3464 function Babel.begin_process_input()
3465   if luatexbase and luatexbase.add_to_callback then
3466     luatexbase.add_to_callback('process_input_buffer',
3467       Babel.bytes, 'Babel.bytes')
3468   else
3469     Babel.callback = callback.find('process_input_buffer')
3470     callback.register('process_input_buffer', Babel.bytes)
3471   end
3472 end
3473 function Babel.end_process_input ()
3474   if luatexbase and luatexbase.remove_from_callback then
3475     luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
3476   else
3477     callback.register('process_input_buffer', Babel.callback)
3478   end
3479 end
3480 function Babel.addpatterns(pp, lg)
3481   local lg = lang.new(lg)
3482   local pats = lang.patterns(lg) or ''
3483   lang.clear_patterns(lg)
3484   for p in pp:gmatch('[^%s]+') do
3485     ss = ''
3486     for i in string.utfcharacters(p:gsub('%d', '')) do
3487       ss = ss .. '%d?' .. i
3488     end
3489     ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
3490     ss = ss:gsub('%.%%d%?$', '%%.')
3491     pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
3492     if n == 0 then
3493       tex.sprint(
3494         [[\string\csname\space bbl@info\endcsname{New pattern: }}
3495         .. p .. [[}}]])
3496       pats = pats .. ' ' .. p
3497     else
3498       tex.sprint(
3499         [[\string\csname\space bbl@info\endcsname{Renew pattern: }}
3500         .. p .. [[}}]])
3501     end
3502   end
3503   lang.patterns(lg, pats)
3504 end
3505 }

```

```

3506 \endgroup
3507 \def\BabelStringsDefault{unicode}
3508 \let\luabbl@stop\relax
3509 \AddBabelHook{luatex}{encodedcommands}{%
3510   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
3511   \ifx\bbl@tempa\bbl@tempb\else
3512     \directlua{Babel.begin_process_input()}%
3513     \def\luabbl@stop{%
3514       \directlua{Babel.end_process_input()}}%
3515   \fi}%
3516 \AddBabelHook{luatex}{stopcommands}{%
3517   \luabbl@stop
3518   \let\luabbl@stop\relax}
3519 \AddBabelHook{luatex}{patterns}{%
3520   \@ifundefined{bbl@hyphendata@the\language}%
3521   {\def\bbl@elt##1##2##3##4{%
3522     \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
3523     \def\bbl@tempb{##3}%
3524     \ifx\bbl@tempb\@empty\else % if not a synonymous
3525       \def\bbl@tempc{##3}{##4}}%
3526     \fi
3527     \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
3528     \fi}%
3529   \bbl@languages
3530   \@ifundefined{bbl@hyphendata@the\language}%
3531   {\bbl@info{No hyphenation patterns were set for\%
3532     language '#2'. Reported}}%
3533   {\expandafter\expandafter\expandafter\bbl@luapatterns
3534     \csname bbl@hyphendata@the\language\endcsname}}}%
3535   \@ifundefined{bbl@patterns@}{}%
3536   \begingroup
3537     \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
3538     \ifin@else
3539       \ifx\bbl@patterns@\@empty\else
3540         \directlua{ Babel.addpatterns(
3541           [[\bbl@patterns@]], \number\language) }%
3542         \fi
3543         \@ifundefined{bbl@patterns@#1}%
3544         {\@empty
3545           {\directlua{ Babel.addpatterns(
3546             [[\space\csname bbl@patterns@#1\endcsname]],
3547             \number\language) }}%
3548           \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
3549         \fi
3550       \endgroup}}
3551 \AddBabelHook{luatex}{everylanguage}{%
3552   \def\process@language##1##2##3{%
3553     \def\process@line####1####2 ####3 ####4 {}}}
3554 \AddBabelHook{luatex}{loadpatterns}{%
3555   \input #1\relax
3556   \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
3557     {#1}{}}
3558 \AddBabelHook{luatex}{loadexceptions}{%
3559   \input #1\relax
3560   \def\bbl@tempb##1##2{##1}{##1}%
3561   \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
3562     {\expandafter\expandafter\expandafter\bbl@tempb
3563       \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.

```

3564 \@onlypreamble\babelpatterns
3565 \AtEndOfPackage{%
3566   \newcommand\babelpatterns[2][\@empty]{%
3567     \ifx\bbl@patterns@relax
3568       \let\bbl@patterns@\@empty
3569     \fi
3570     \ifx\bbl@pttnlist\@empty\else
3571       \bbl@warning{%
3572         You must not intermingle \string\selectlanguage\space and\%
3573         \string\babelpatterns\space or some patterns will not\%
3574         be taken into account. Reported}%
3575       \fi
3576       \ifx\@empty#1%
3577         \protected@edef\bbl@patterns@\{\bbl@patterns@\space#2}%
3578       \else
3579         \edef\bbl@tempb{\zap@space#1 \@empty}%
3580         \bbl@for\bbl@tempa\bbl@tempb{%
3581           \bbl@fixname\bbl@tempa
3582           \bbl@iflanguage\bbl@tempa{%
3583             \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
3584               \@ifundefined{bbl@patterns@\bbl@tempa}%
3585                 \@empty
3586                 {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
3587               #2}}}%
3588         \fi}}

```

Common stuff.

```

3589 \AddBabelHook{luatex}{loadkernel}{%
3590   <<Restore Unicode catcodes before loading patterns>>}}
3591 \ifx\DisableBabelHook\@undefined\endinput\fi
3592 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
3593 \DisableBabelHook{babel-fontspec}
3594 <<Font selection>>

```

## 14.4 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`.

```

3595 \bbl@trace{Redefinitions for bidi layout}
3596 \ifx\@eqnnum\@undefined\else
3597   \ifx\bbl@attr@dir\@undefined\else
3598     \edef\@eqnnum{%
3599       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
3600       \unexpanded\expandafter{\@eqnnum}}}}
3601 \fi
3602 \fi

```

```

3603 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
3604 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
3605 \def\bbl@nextfake#1{%
3606   \mathdir\bodydir % non-local, use always inside a group!
3607   \bbl@exp{%
3608     #1%           Once entered in math, set boxes to restore values
3609     \everyvbox{%
3610       \the\everyvbox
3611       \bodydir\the\bodydir
3612       \mathdir\the\mathdir
3613       \everyhbox{\the\everyhbox}%
3614       \everyvbox{\the\everyvbox}}%
3615     \everyhbox{%
3616       \the\everyhbox
3617       \bodydir\the\bodydir
3618       \mathdir\the\mathdir
3619       \everyhbox{\the\everyhbox}%
3620       \everyvbox{\the\everyvbox}}}%
3621 \def\@hangfrom#1{%
3622   \setbox\@tempboxa\hbox{{#1}}%
3623   \hangindent\wd\@tempboxa
3624   \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
3625     \shapemode\@ne
3626   \fi
3627   \noindent\box\@tempboxa}
3628 \fi
3629 \IfBabelLayout{tabular}
3630 {\def\@tabular{%
3631   \leavevmode\hbox\bgroup\bbl@nextfake$% %$
3632   \let\@acol\@tabacol \let\@classz\@tabclassz
3633   \let\@classiv\@tabclassiv \let\@tabularcr\@tabarray}}
3634 {}
3635 \IfBabelLayout{lists}
3636 {\def\list#1#2{%
3637   \ifnum \@listdepth >5\relax
3638     \@toodeep
3639   \else
3640     \global\advance\@listdepth\@ne
3641   \fi
3642   \rightmargin\z@
3643   \listparindent\z@
3644   \itemindent\z@
3645   \csname @list\romannumeral\the\@listdepth\endcsname
3646   \def\@itemlabel{#1}%
3647   \let\makelabel\@mklab
3648   \@nmbrlistfalse
3649   #2\relax
3650   \@trivlist
3651   \parskip\parsep
3652   \parindent\listparindent
3653   \advance\linewidth -\rightmargin
3654   \advance\linewidth -\leftmargin
3655   \advance\@totalleftmargin \leftmargin
3656   \parshape \@ne
3657   \@totalleftmargin \linewidth
3658   \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
3659     \shapemode\tw@
3660   \fi
3661   \ignorespaces}}

```



```
3662 {}
```

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.

```
3663 \IfBabelLayout{counters}%
3664 {\def\@textsuperscript#1{% lua has separate settings for math
3665   \m@th
3666   \mathdir\pagedir % required with basic-r; ok with default, too
3667   \ensuremath{^{\mbox{\fontsize \sf@size \z@ #1}}}%
3668   \let\bbl@latinarabic=\@arabic
3669   \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
3670   \@ifpackagewith{babel}{bidi=default}%
3671   {\let\bbl@asciroman=\@roman
3672    \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
3673    \let\bbl@asciiRoman=\@Roman
3674    \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
3675    \def\labelenumii{}\theenumii}%
3676    \def\p@enumiii{\p@enumii}\theenumii{}\}}{}
3677 \langle Footnote changes \rangle
3678 \IfBabelLayout{footnotes}%
3679 {\BabelFootnote\footnote\language\language{}{}}%
3680 \BabelFootnote\localfootnote\language\language{}{}}%
3681 \BabelFootnote\mainfootnote{}{}}{}
3682 {}
```

Some  $\text{\LaTeX}$  macros use internally the math mode for text formatting. They have very little in common and are grouped here, as a single option.

```
3683 \IfBabelLayout{extras}%
3684 {\def\@underline#1{%
3685   \relax
3686   \ifmmode\@underline{#1}%
3687   \else\bbl@nextfake$@@underline{\hbox{#1}}\m@th$\relax\fi}%
3688 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
3689   \if b\expandafter\car\@series\@nil\boldmath\fi
3690   \babelsublr{%
3691     \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\mathrel{\varepsilon}}$}}}
3692 {}
3693 \langle /luatex \rangle
```

## 14.5 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.

TODO: math mode (as weak L?)

```

3694 (*basic-r)
3695 Babel = Babel or {}
3696
3697 require('babel-bidi.lua')
3698
3699 local characters = Babel.characters
3700 local ranges = Babel.ranges
3701
3702 local DIR = node.id("dir")
3703
3704 local function dir_mark(head, from, to, outer)
3705   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
3706   local d = node.new(DIR)
3707   d.dir = '+' .. dir
3708   node.insert_before(head, from, d)
3709   d = node.new(DIR)
3710   d.dir = '-' .. dir
3711   node.insert_after(head, to, d)
3712 end
3713
3714 function Babel.pre_otfload_v(head)
3715   -- head = Babel.numbers(head)
3716   head = Babel.bidi(head, true)
3717   return head
3718 end
3719
3720 function Babel.pre_otfload_h(head)
3721   -- head = Babel.numbers(head)
3722   head = Babel.bidi(head, false)
3723   return head
3724 end
3725
3726 function Babel.bidi(head, ispar)
3727   local first_n, last_n          -- first and last char with nums
3728   local last_es                  -- an auxiliary 'last' used with nums
3729   local first_d, last_d          -- first and last char in L/R block
3730   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):

```

3731 local strong = ('TRT' == tex.pardir) and 'r' or 'l'
3732 local strong_lr = (strong == 'l') and 'l' or 'r'

```

```

3733 local outer = strong
3734
3735 local new_dir = false
3736 local first_dir = false
3737
3738 local last_lr
3739
3740 local type_n = ''
3741
3742 for item in node.traverse(head) do
3743
3744   -- three cases: glyph, dir, otherwise
3745   if item.id == node.id'glyph'
3746     or (item.id == 7 and item.subtype == 2) then
3747
3748     local itemchar
3749     if item.id == 7 and item.subtype == 2 then
3750       itemchar = item.replace.char
3751     else
3752       itemchar = item.char
3753     end
3754     local chardata = characters[itemchar]
3755     dir = chardata and chardata.d or nil
3756     if not dir then
3757       for nn, et in ipairs(ranges) do
3758         if itemchar < et[1] then
3759           break
3760         elseif itemchar <= et[2] then
3761           dir = et[3]
3762           break
3763         end
3764       end
3765     end
3766     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.

```

3767   if new_dir then
3768     attr_dir = 0
3769     for at in node.traverse(item.attr) do
3770       if at.number == luatexbase.registernumber'bbl@attr@dir' then
3771         attr_dir = at.value % 3
3772       end
3773     end
3774     if attr_dir == 1 then
3775       strong = 'r'
3776     elseif attr_dir == 2 then
3777       strong = 'al'
3778     else
3779       strong = 'l'
3780     end
3781     strong_lr = (strong == 'l') and 'l' or 'r'
3782     outer = strong_lr
3783     new_dir = false
3784   end
3785
3786   if dir == 'nsm' then dir = strong end

```

-- W1

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

```
3787     dir_real = dir          -- We need dir_real to set strong below
3788     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:

```
3789     if strong == 'al' then
3790         if dir == 'en' then dir = 'an' end          -- W2
3791         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
3792         strong_lr = 'r'                             -- W3
3793     end
```

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

```
3794     elseif item.id == node.id'dir' then
3795         new_dir = true
3796         dir = nil
3797     else
3798         dir = nil          -- Not a char
3799     end
```

Numbers in R mode. A sequence of <en>, <et>, <an>, <es> and <cs> is typeset (with some rules) in L mode. We store the starting and ending points, and only when anything different is found (including nil, ie, a non-char), the textdir is set. This means you cannot insert, say, a whatsit, but this is what I would expect (with luacolor you may colorize some digits). Anyway, this behavior could be changed with a switch in the future. Note in the first branch only <an> is relevant if <al>.

```
3800     if dir == 'en' or dir == 'an' or dir == 'et' then
3801         if dir ~= 'et' then
3802             type_n = dir
3803         end
3804         first_n = first_n or item
3805         last_n = last_es or item
3806         last_es = nil
3807     elseif dir == 'es' and last_n then -- W3+W6
3808         last_es = item
3809     elseif dir == 'cs' then          -- it's right - do nothing
3810     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
3811         if strong_lr == 'r' and type_n ~= '' then
3812             dir_mark(head, first_n, last_n, 'r')
3813         elseif strong_lr == 'l' and first_d and type_n == 'an' then
3814             dir_mark(head, first_n, last_n, 'r')
3815             dir_mark(head, first_d, last_d, outer)
3816             first_d, last_d = nil, nil
3817         elseif strong_lr == 'l' and type_n ~= '' then
3818             last_d = last_n
3819         end
3820         type_n = ''
3821         first_n, last_n = nil, nil
3822     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:

```
3823     if dir == 'l' or dir == 'r' then
3824         if dir ~= outer then
3825             first_d = first_d or item
```

```

3826     last_d = item
3827     elseif first_d and dir ~= strong_lr then
3828         dir_mark(head, first_d, last_d, outer)
3829         first_d, last_d = nil, nil
3830     end
3831 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.

```

3832     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
3833         item.char = characters[item.char] and
3834             characters[item.char].m or item.char
3835     elseif (dir or new_dir) and last_lr ~= item then
3836         local mir = outer .. strong_lr .. (dir or outer)
3837         if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
3838             for ch in node.traverse(node.next(last_lr)) do
3839                 if ch == item then break end
3840                 if ch.id == node.id'glyph' then
3841                     ch.char = characters[ch.char].m or ch.char
3842                 end
3843             end
3844         end
3845     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).

```

3846     if dir == 'l' or dir == 'r' then
3847         last_lr = item
3848         strong = dir_real -- Don't search back - best save now
3849         strong_lr = (strong == 'l') and 'l' or 'r'
3850     elseif new_dir then
3851         last_lr = nil
3852     end
3853 end

```

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

```

3854     if last_lr and outer == 'r' then
3855         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
3856             ch.char = characters[ch.char].m or ch.char
3857         end
3858     end
3859     if first_n then
3860         dir_mark(head, first_n, last_n, outer)
3861     end
3862     if first_d then
3863         dir_mark(head, first_d, last_d, outer)
3864     end

```

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

```

3865     return node.prev(head) or head
3866 end
3867 </basic-r>

```

And here the Lua code for bidi=basic:

```

3868 <*basic>

```

```

3869 Babel = Babel or {}
3870
3871 Babel.fontmap = Babel.fontmap or {}
3872 Babel.fontmap[0] = {}      -- l
3873 Babel.fontmap[1] = {}      -- r
3874 Babel.fontmap[2] = {}      -- al/an
3875
3876 function Babel.pre_otfload_v(head)
3877   -- head = Babel.numbers(head)
3878   head = Babel.bidi(head, true)
3879   return head
3880 end
3881
3882 function Babel.pre_otfload_h(head, gc, sz, pt, dir)
3883   -- head = Babel.numbers(head)
3884   head = Babel.bidi(head, false, dir)
3885   return head
3886 end
3887
3888 require('babel-bidi.lua')
3889
3890 local characters = Babel.characters
3891 local ranges = Babel.ranges
3892
3893 local DIR = node.id('dir')
3894 local GLYPH = node.id('glyph')
3895
3896 local function insert_implicit(head, state, outer)
3897   local new_state = state
3898   if state.sim and state.eim and state.sim ~= state.eim then
3899     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
3900     local d = node.new(DIR)
3901     d.dir = '+' .. dir
3902     node.insert_before(head, state.sim, d)
3903     local d = node.new(DIR)
3904     d.dir = '-' .. dir
3905     node.insert_after(head, state.eim, d)
3906   end
3907   new_state.sim, new_state.eim = nil, nil
3908   return head, new_state
3909 end
3910
3911 local function insert_numeric(head, state)
3912   local new
3913   local new_state = state
3914   if state.san and state.ean and state.san ~= state.ean then
3915     local d = node.new(DIR)
3916     d.dir = '+TLT'
3917     _, new = node.insert_before(head, state.san, d)
3918     if state.san == state.sim then state.sim = new end
3919     local d = node.new(DIR)
3920     d.dir = '-TLT'
3921     _, new = node.insert_after(head, state.ean, d)
3922     if state.ean == state.eim then state.eim = new end
3923   end
3924   new_state.san, new_state.ean = nil, nil
3925   return head, new_state
3926 end
3927

```

```

3928 -- \hbox with an explicit dir can lead to wrong results
3929 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>
3930
3931 function Babel.bidi(head, ispar, hdir)
3932   local d -- d is used mainly for computations in a loop
3933   local prev_d = ''
3934   local new_d = false
3935
3936   local nodes = {}
3937   local outer_first = nil
3938
3939   local glue_d = nil
3940   local glue_i = nil
3941
3942   local has_en = false
3943   local first_et = nil
3944
3945   local ATDIR = luatexbase.registernumber'bbl@attr@dir'
3946
3947   local save_outer
3948   local temp = node.get_attribute(head, ATDIR)
3949   if temp then
3950     temp = temp % 3
3951     save_outer = (temp == 0 and 'l') or
3952                  (temp == 1 and 'r') or
3953                  (temp == 2 and 'al')
3954   elseif ispar then -- Or error? Shouldn't happen
3955     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
3956   else
3957     save_outer = ('TRT' == hdir) and 'r' or 'l'
3958   end
3959   local outer = save_outer
3960   local last = outer
3961   -- 'al' is only taken into account in the first, current loop
3962   if save_outer == 'al' then save_outer = 'r' end
3963
3964   local fontmap = Babel.fontmap
3965
3966   for item in node.traverse(head) do
3967
3968     -- In what follows, #node is the last (previous) node, because the
3969     -- current one is not added until we start processing the neutrals.
3970
3971     -- three cases: glyph, dir, otherwise
3972     if item.id == GLYPH
3973       or (item.id == 7 and item.subtype == 2) then
3974
3975       local d_font = nil
3976       local item_r
3977       if item.id == 7 and item.subtype == 2 then
3978         item_r = item.replace -- automatic discs have just 1 glyph
3979       else
3980         item_r = item
3981       end
3982       local chardata = characters[item_r.char]
3983       d = chardata and chardata.d or nil
3984       if not d or d == 'nsm' then
3985         for nn, et in ipairs(ranges) do
3986           if item_r.char < et[1] then

```

```

3987         break
3988     elseif item_r.char <= et[2] then
3989         if not d then d = et[3]
3990         elseif d == 'nsm' then d_font = et[3]
3991         end
3992         break
3993     end
3994 end
3995 end
3996 d = d or 'l'
3997 d_font = d_font or d
3998
3999 d_font = (d_font == 'l' and 0) or
4000         (d_font == 'nsm' and 0) or
4001         (d_font == 'r' and 1) or
4002         (d_font == 'al' and 2) or
4003         (d_font == 'an' and 2) or nil
4004 if d_font and fontmap and fontmap[d_font][item_r.font] then
4005     item_r.font = fontmap[d_font][item_r.font]
4006 end
4007
4008 if new_d then
4009     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
4010     attr_d = node.get_attribute(item, ATDIR)
4011     attr_d = attr_d % 3
4012     if attr_d == 1 then
4013         outer_first = 'r'
4014         last = 'r'
4015     elseif attr_d == 2 then
4016         outer_first = 'r'
4017         last = 'al'
4018     else
4019         outer_first = 'l'
4020         last = 'l'
4021     end
4022     outer = last
4023     has_en = false
4024     first_et = nil
4025     new_d = false
4026 end
4027
4028 if glue_d then
4029     if (d == 'l' and 'l' or 'r') ~= glue_d then
4030         table.insert(nodes, {glue_i, 'on', nil})
4031     end
4032     glue_d = nil
4033     glue_i = nil
4034 end
4035
4036 elseif item.id == DIR then
4037     d = nil
4038     new_d = true
4039
4040 elseif item.id == node.id'glue' and item.subtype == 13 then
4041     glue_d = d
4042     glue_i = item
4043     d = nil
4044
4045 else

```



```

4046     d = nil
4047 end
4048
4049 -- AL <= EN/ET/ES      -- W2 + W3 + W6
4050 if last == 'al' and d == 'en' then
4051     d = 'an'           -- W3
4052 elseif last == 'al' and (d == 'et' or d == 'es') then
4053     d = 'on'           -- W6
4054 end
4055
4056 -- EN + CS/ES + EN      -- W4
4057 if d == 'en' and #nodes >= 2 then
4058     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
4059         and nodes[#nodes-1][2] == 'en' then
4060         nodes[#nodes][2] = 'en'
4061     end
4062 end
4063
4064 -- AN + CS + AN         -- W4 too, because uax9 mixes both cases
4065 if d == 'an' and #nodes >= 2 then
4066     if (nodes[#nodes][2] == 'cs')
4067         and nodes[#nodes-1][2] == 'an' then
4068         nodes[#nodes][2] = 'an'
4069     end
4070 end
4071
4072 -- ET/EN                -- W5 + W7->1 / W6->on
4073 if d == 'et' then
4074     first_et = first_et or (#nodes + 1)
4075 elseif d == 'en' then
4076     has_en = true
4077     first_et = first_et or (#nodes + 1)
4078 elseif first_et then    -- d may be nil here !
4079     if has_en then
4080         if last == 'l' then
4081             temp = 'l'    -- W7
4082         else
4083             temp = 'en'   -- W5
4084         end
4085     else
4086         temp = 'on'       -- W6
4087     end
4088     for e = first_et, #nodes do
4089         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
4090     end
4091     first_et = nil
4092     has_en = false
4093 end
4094
4095 if d then
4096     if d == 'al' then
4097         d = 'r'
4098         last = 'al'
4099     elseif d == 'l' or d == 'r' then
4100         last = d
4101     end
4102     prev_d = d
4103     table.insert(nodes, {item, d, outer_first})
4104 end

```

```

4105
4106     outer_first = nil
4107
4108 end
4109
4110 -- TODO -- repeated here in case EN/ET is the last node. Find a
4111 -- better way of doing things:
4112 if first_et then      -- dir may be nil here !
4113     if has_en then
4114         if last == 'l' then
4115             temp = 'l'    -- W7
4116         else
4117             temp = 'en'   -- W5
4118         end
4119     else
4120         temp = 'on'       -- W6
4121     end
4122     for e = first_et, #nodes do
4123         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
4124     end
4125 end
4126
4127 -- dummy node, to close things
4128 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
4129
4130 ----- NEUTRAL -----
4131
4132 outer = save_outer
4133 last = outer
4134
4135 local first_on = nil
4136
4137 for q = 1, #nodes do
4138     local item
4139
4140     local outer_first = nodes[q][3]
4141     outer = outer_first or outer
4142     last = outer_first or last
4143
4144     local d = nodes[q][2]
4145     if d == 'an' or d == 'en' then d = 'r' end
4146     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
4147
4148     if d == 'on' then
4149         first_on = first_on or q
4150     elseif first_on then
4151         if last == d then
4152             temp = d
4153         else
4154             temp = outer
4155         end
4156         for r = first_on, q - 1 do
4157             nodes[r][2] = temp
4158             item = nodes[r][1]    -- MIRRORING
4159             if item.id == GLYPH and temp == 'r' then
4160                 item.char = characters[item.char].m or item.char
4161             end
4162         end
4163         first_on = nil

```

```

4164     end
4165
4166     if d == 'r' or d == 'l' then last = d end
4167 end
4168
4169 ----- IMPLICIT, REORDER -----
4170
4171 outer = save_outer
4172 last = outer
4173
4174 local state = {}
4175 state.has_r = false
4176
4177 for q = 1, #nodes do
4178
4179     local item = nodes[q][1]
4180
4181     outer = nodes[q][3] or outer
4182
4183     local d = nodes[q][2]
4184
4185     if d == 'nsm' then d = last end          -- W1
4186     if d == 'en' then d = 'an' end
4187     local isdir = (d == 'r' or d == 'l')
4188
4189     if outer == 'l' and d == 'an' then
4190         state.san = state.san or item
4191         state.ean = item
4192     elseif state.san then
4193         head, state = insert_numeric(head, state)
4194     end
4195
4196     if outer == 'l' then
4197         if d == 'an' or d == 'r' then      -- im -> implicit
4198             if d == 'r' then state.has_r = true end
4199             state.sim = state.sim or item
4200             state.eim = item
4201         elseif d == 'l' and state.sim and state.has_r then
4202             head, state = insert_implicit(head, state, outer)
4203         elseif d == 'l' then
4204             state.sim, state.eim, state.has_r = nil, nil, false
4205         end
4206     else
4207         if d == 'an' or d == 'l' then
4208             state.sim = state.sim or item
4209             state.eim = item
4210         elseif d == 'r' and state.sim then
4211             head, state = insert_implicit(head, state, outer)
4212         elseif d == 'r' then
4213             state.sim, state.eim = nil, nil
4214         end
4215     end
4216
4217     if isdir then
4218         last = d          -- Don't search back - best save now
4219     elseif d == 'on' and state.san then
4220         state.san = state.san or item
4221         state.ean = item
4222     end

```

```

4223
4224   end
4225
4226   return node.prev(head) or head
4227 end
4228 </basic>

```

## 15 The ‘nil’ language

This ‘language’ does nothing, except setting the hyphenation patterns to nohyphenation. For this language currently no special definitions are needed or available. The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```

4229 <*nil>
4230 \ProvidesLanguage{nil}[<<date>> <<version>> Nil language]
4231 \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.

```

4232 \ifx\l@nohyphenation\@undefined
4233   \@nopatterns{nil}
4234   \adddialect\l@nil0
4235 \else
4236   \let\l@nil\l@nohyphenation
4237 \fi

```

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

```

4238 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

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

```

\captionnil
\datenil
4239 \let\captionnil\@empty
4240 \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.

```

4241 \ldf@finish{nil}
4242 </nil>

```

## 16 Support for Plain $\text{\TeX}$ (plain.def)

### 16.1 Not renaming `hyphen.tex`

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based  $\text{\TeX}$ -format. When asked he responded:

That file name is “sacred”, and if anybody changes it they will cause severe upward/downward compatibility headaches.

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

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to achieve the desired effect, based on the `babel` package. If you load each of them with `iniTeX`, you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`. As these files are going to be read as the first thing `iniTeX` sees, we need to set some category codes just to be able to change the definition of `\input`

```
4243 (*bplain | blplain)
4244 \catcode`\{=1 % left brace is begin-group character
4245 \catcode`\}=2 % right brace is end-group character
4246 \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...

```
4247 \openin 0 hyphen.cfg
```

If the file wasn't found the following test turns out true.

```
4248 \ifeof0
4249 \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).

```
4250 \let\input
```

Then `\input` is defined to forget about its argument and load `hyphen.cfg` instead.

```
4251 \def\input #1 {%
4252   \let\input\input
4253   \input hyphen.cfg
```

Once that's done the original meaning of `\input` can be restored and the definition of `\input` can be forgotten.

```
4254   \let\input\undefined
4255 }
4256 \fi
4257 (/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`.

```
4258 (bplain)\input plain.tex
4259 (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.

```
4260 (bplain)\def\fmtname{babel-plain}
4261 (blplain)\def\fmtname{babel-lplain}
```

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

## 16.2 Emulating some $\text{\LaTeX}$ features

The following code duplicates or emulates parts of  $\text{\LaTeX} 2_{\epsilon}$  that are needed for `babel`.

```
4262 (*plain)
4263 \def\@empty{}
4264 \def\loadlocalcfg#1{%
```

```

4265 \openin0#1.cfg
4266 \ifeof0
4267 \closein0
4268 \else
4269 \closein0
4270 {\immediate\write16{*****}%
4271 \immediate\write16{* Local config file #1.cfg used}%
4272 \immediate\write16{*}%
4273 }
4274 \input #1.cfg\relax
4275 \fi
4276 \@endoflfd}

```

## 16.3 General tools

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

```

4277 \long\def\@firstofone#1{#1}
4278 \long\def\@firstoftwo#1#2{#1}
4279 \long\def\@secondoftwo#1#2{#2}
4280 \def\@nnil{\@nil}
4281 \def\@gobbletwo#1#2{}
4282 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
4283 \def\@star@or@long#1{%
4284 \@ifstar
4285 {\let\l@ngrel@x\relax#1}%
4286 {\let\l@ngrel@x\long#1}}
4287 \let\l@ngrel@x\relax
4288 \def\@car#1#2\@nil{#1}
4289 \def\@cdr#1#2\@nil{#2}
4290 \let\@typeset@protect\relax
4291 \let\protected@edef\edef
4292 \long\def\@gobble#1{}
4293 \edef\@backslashchar{\expandafter\@gobble\string\}
4294 \def\strip@prefix#1>{}
4295 \def\g@addto@macro#1#2{%
4296 \toks@{\expandafter{#1#2}}%
4297 \xdef#1{\the\toks@}}
4298 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
4299 \def\@nameuse#1{\csname #1\endcsname}
4300 \def\@ifundefined#1{%
4301 \expandafter\ifx\csname#1\endcsname\relax
4302 \expandafter\@firstoftwo
4303 \else
4304 \expandafter\@secondoftwo
4305 \fi}
4306 \def\@expandtwoargs#1#2#3{%
4307 \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
4308 \def\zap@space#1 #2{%
4309 #1%
4310 \ifx#2\@empty\else\expandafter\zap@space\fi
4311 #2}

```

$\text{\LaTeX}$  2<sub>ε</sub> has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

4312 \ifx\@preamblecmds\@undefined
4313 \def\@preamblecmds{}
4314 \fi
4315 \def\@onlypreamble#1{%

```

```

4316 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
4317   \@preamblecmds\do#1}}
4318 \@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.

```

4319 \def\begindocument{%
4320   \@begindocumenthook
4321   \global\let\@begindocumenthook\@undefined
4322   \def\do#1{\global\let##1\@undefined}%
4323   \@preamblecmds
4324   \global\let\do\noexpand}

4325 \ifx\@begindocumenthook\@undefined
4326   \def\@begindocumenthook{}
4327 \fi
4328 \@onlypreamble\@begindocumenthook
4329 \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.

```

4330 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
4331 \@onlypreamble\AtEndOfPackage
4332 \def\@endofldf{}
4333 \@onlypreamble\@endofldf
4334 \let\bbl@afterlang\@empty
4335 \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.

```

4336 \ifx\if@files\@undefined
4337   \expandafter\let\csname if@files\expandafter\endcsname
4338   \csname iffalse\endcsname
4339 \fi

```

Mimick L<sup>A</sup>T<sub>E</sub>X's commands to define control sequences.

```

4340 \def\newcommand{\@star@or@long\new@command}
4341 \def\new@command#1{%
4342   \@testopt{\@newcommand#1}0}
4343 \def\@newcommand#1[#2]{%
4344   \@ifnextchar [{\@xargdef#1[#2]}%
4345               {\@argdef#1[#2]}}
4346 \long\def\@argdef#1[#2]#3{%
4347   \@yargdef#1\@ne{#2}{#3}}
4348 \long\def\@xargdef#1[#2][#3]#4{%
4349   \expandafter\def\expandafter#1\expandafter{%
4350     \expandafter\@protected@testopt\expandafter #1%
4351     \csname\string#1\expandafter\endcsname{#3}}%
4352   \expandafter\@yargdef \csname\string#1\endcsname
4353   \tw@{#2}{#4}}
4354 \long\def\@yargdef#1#2#3{%
4355   \@tempcnta#3\relax
4356   \advance \@tempcnta \@ne
4357   \let\@hash@\relax
4358   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
4359   \@tempcntb #2%
4360   \@whilenum\@tempcntb <\@tempcnta
4361   \do{%
4362     \edef\reserved@a{\reserved@a\@hash@the\@tempcntb}%

```

```

4363 \advance\@tempcntb \@ne}%
4364 \let\@hash@##%
4365 \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
4366 \def\providecommand{\@star@or@long\provide@command}
4367 \def\provide@command#1{%
4368 \begingroup
4369 \escapechar\m@ne\xdef\@gtempa{\string#1}%
4370 \endgroup
4371 \expandafter\@ifundefined\@gtempa
4372 {\def\reserved@a{\new@command#1}}%
4373 {\let\reserved@a\relax
4374 \def\reserved@a{\new@command\reserved@a}}%
4375 \reserved@a}%

4376 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
4377 \def\declare@robustcommand#1{%
4378 \edef\reserved@a{\string#1}%
4379 \def\reserved@b{#1}%
4380 \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
4381 \edef#1{%
4382 \ifx\reserved@a\reserved@b
4383 \noexpand\x@protect
4384 \noexpand#1%
4385 \fi
4386 \noexpand\protect
4387 \expandafter\noexpand\csname
4388 \expandafter\@gobble\string#1 \endcsname
4389 }%
4390 \expandafter\new@command\csname
4391 \expandafter\@gobble\string#1 \endcsname
4392 }
4393 \def\x@protect#1{%
4394 \ifx\protect\@typeset@protect\else
4395 \@x@protect#1%
4396 \fi
4397 }
4398 \def\@x@protect#1\fi#2#3{%
4399 \fi\protect#1%
4400 }

```

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`.

```

4401 \def\bbl@tempa{\csname newif\endcsname\in@}
4402 \ifx\in@\@undefined
4403 \def\in@#1#2{%
4404 \def\in@##1#1##2##3\in@{%
4405 \ifx\in@##2\in@false\else\in@true\fi}%
4406 \in@#2#1\in@\in@}
4407 \else
4408 \let\bbl@tempa\@empty
4409 \fi
4410 \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).

```
4411 \def\ifpackagewith#1#2#3#4{#3}
```

The  $\LaTeX$  macro `\ifl@aded` checks whether a file was loaded. This functionality is not needed for plain  $\TeX$  but we need the macro to be defined as a no-op.

```
4412 \def\ifl@aded#1#2#3#4{}
```

For the following code we need to make sure that the commands `\newcommand` and `\providecommand` exist with some sensible definition. They are not fully equivalent to their  $\LaTeX$  2<sub>ε</sub> versions; just enough to make things work in plain  $\TeX$  environments.

```
4413 \ifx\@tempcnta\@undefined
4414   \csname newcount\endcsname\@tempcnta\relax
4415 \fi
4416 \ifx\@tempcntb\@undefined
4417   \csname newcount\endcsname\@tempcntb\relax
4418 \fi
```

To prevent wasting two counters in  $\LaTeX$  2.09 (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```
4419 \ifx\bye\@undefined
4420   \advance\count10 by -2\relax
4421 \fi
4422 \ifx\@ifnextchar\@undefined
4423   \def\@ifnextchar#1#2#3{%
4424     \let\reserved@d=#1%
4425     \def\reserved@a{#2}\def\reserved@b{#3}%
4426     \futurelet\@let@token\@ifnch}
4427   \def\@ifnch{%
4428     \ifx\@let@token\@sptoken
4429       \let\reserved@c\@xifnch
4430     \else
4431       \ifx\@let@token\reserved@d
4432         \let\reserved@c\reserved@a
4433       \else
4434         \let\reserved@c\reserved@b
4435     \fi
4436   \fi
4437   \reserved@c}
4438   \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
4439   \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
4440 \fi
4441 \def\@testopt#1#2{%
4442   \@ifnextchar[#{1}{#1[#2]}}
4443 \def\@protected@testopt#1{%
4444   \ifx\protect\@typeset@protect
4445     \expandafter\@testopt
4446   \else
4447     \@x@protect#1%
4448   \fi}
4449 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
4450   #2\relax}\fi}
4451 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
4452   \else\expandafter\@gobble\fi{#1}}
```

## 16.4 Encoding related macros

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

```

4453 \def\DeclareTextCommand{%
4454   \@dec@text@cmd\providecommand
4455 }
4456 \def\ProvideTextCommand{%
4457   \@dec@text@cmd\providecommand
4458 }
4459 \def\DeclareTextSymbol#1#2#3{%
4460   \@dec@text@cmd\chardef#1{#2}#3\relax
4461 }
4462 \def\@dec@text@cmd#1#2#3{%
4463   \expandafter\def\expandafter#2%
4464     \expandafter{%
4465       \csname#3-cmd\expandafter\endcsname
4466       \expandafter#2%
4467       \csname#3\string#2\endcsname
4468     }%
4469 %   \let\@ifdefinable\@rc@ifdefinable
4470   \expandafter#1\csname#3\string#2\endcsname
4471 }
4472 \def\@current@cmd#1{%
4473   \ifx\protect\@typeset@protect\else
4474     \noexpand#1\expandafter\@gobble
4475   \fi
4476 }
4477 \def\@changed@cmd#1#2{%
4478   \ifx\protect\@typeset@protect
4479     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
4480       \expandafter\ifx\csname ?\string#1\endcsname\relax
4481         \expandafter\def\csname ?\string#1\endcsname{%
4482           \@changed@x@err{#1}%
4483         }%
4484       \fi
4485       \global\expandafter\let
4486         \csname\cf@encoding \string#1\expandafter\endcsname
4487         \csname ?\string#1\endcsname
4488     \fi
4489     \csname\cf@encoding\string#1%
4490     \expandafter\endcsname
4491   \else
4492     \noexpand#1%
4493   \fi
4494 }
4495 \def\@changed@x@err#1{%
4496   \errhelp{Your command will be ignored, type <return> to proceed}%
4497   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
4498 \def\DeclareTextCommandDefault#1{%
4499   \DeclareTextCommand#1?%
4500 }
4501 \def\ProvideTextCommandDefault#1{%
4502   \ProvideTextCommand#1?%
4503 }
4504 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
4505 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
4506 \def\DeclareTextAccent#1#2#3{%
4507   \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
4508 }
4509 \def\DeclareTextCompositeCommand#1#2#3#4{%
4510   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
4511   \edef\reserved@b{\string#1}%

```

```

4512 \edef\reserved@c{%
4513   \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
4514 \ifx\reserved@b\reserved@c
4515   \expandafter\expandafter\expandafter\ifx
4516     \expandafter\@car\reserved@a\relax\relax\@nil
4517     \@text@composite
4518   \else
4519     \edef\reserved@b##1{%
4520       \def\expandafter\noexpand
4521         \csname#2\string#1\endcsname####1{%
4522         \noexpand\@text@composite
4523         \expandafter\noexpand\csname#2\string#1\endcsname
4524         ####1\noexpand\@empty\noexpand\@text@composite
4525         {##1}%
4526       }%
4527     }%
4528     \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
4529   \fi
4530   \expandafter\def\csname\expandafter\string\csname
4531     #2\endcsname\string#1-\string#3\endcsname{#4}
4532 \else
4533   \errhelp{Your command will be ignored, type <return> to proceed}%
4534   \errmessage{\string\DeclareTextCompositeCommand\space used on
4535     inappropriate command \protect#1}
4536 \fi
4537 }
4538 \def\@text@composite#1#2#3\@text@composite{%
4539   \expandafter\@text@composite@x
4540     \csname\string#1-\string#2\endcsname
4541 }
4542 \def\@text@composite@x#1#2{%
4543   \ifx#1\relax
4544     #2%
4545   \else
4546     #1%
4547   \fi
4548 }
4549 %
4550 \def\@strip@args#1:#2-#3\@strip@args{#2}
4551 \def\DeclareTextComposite#1#2#3#4{%
4552   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
4553   \bgroup
4554     \lccode\@=#4%
4555     \lowercase{%
4556   \egroup
4557     \reserved@a @%
4558   }%
4559 }
4560 %
4561 \def\UseTextSymbol#1#2{%
4562 %   \let\@curr@enc\cf@encoding
4563 %   \@use@text@encoding{#1}%
4564 %   #2%
4565 %   \@use@text@encoding\@curr@enc
4566 }
4567 \def\UseTextAccent#1#2#3{%
4568 %   \let\@curr@enc\cf@encoding
4569 %   \@use@text@encoding{#1}%
4570 %   #2{\@use@text@encoding\@curr@enc\selectfont#3}%

```

```

4571 % \use@text@encoding\@curr@enc
4572 }
4573 \def\use@text@encoding#1{%
4574 % \edef\font@encoding{#1}%
4575 % \xdef\font@name{%
4576 % \csname\curr@fontshape/\font@size\endcsname
4577 % }%
4578 % \pickup@font
4579 % \font@name
4580 % \@@enc@update
4581 }
4582 \def\DeclareTextSymbolDefault#1#2{%
4583 \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
4584 }
4585 \def\DeclareTextAccentDefault#1#2{%
4586 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
4587 }
4588 \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.

```

4589 \DeclareTextAccent{"}{OT1}{127}
4590 \DeclareTextAccent{'}{OT1}{19}
4591 \DeclareTextAccent{^}{OT1}{94}
4592 \DeclareTextAccent{\`}{OT1}{18}
4593 \DeclareTextAccent{~}{OT1}{126}

```

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

```

4594 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
4595 \DeclareTextSymbol{\textquotedblright}{OT1}{\`"}
4596 \DeclareTextSymbol{\textquoteleft}{OT1}{\`'}
4597 \DeclareTextSymbol{\textquoteright}{OT1}{\`'}
4598 \DeclareTextSymbol{\i}{OT1}{16}
4599 \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`.

```

4600 \ifx\scriptsize\@undefined
4601 \let\scriptsize\sevenrm
4602 \fi
4603 \</plain>

```

## 17 Acknowledgements

I would like to thank all who volunteered as  $\beta$ -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

## References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.

- [2] 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.