

# Babel

Version 3.43.1993  
2020/04/30

*Original author*  
Johannes L. Braams

*Current maintainer*  
Javier Bezos

Localization and  
internationalization

Unicode

T<sub>E</sub>X

pdfT<sub>E</sub>X

LuaT<sub>E</sub>X

XeT<sub>E</sub>X

# Contents

<b>I</b>	<b>User guide</b>	<b>4</b>
<b>1</b>	<b>The user interface</b>	<b>4</b>
1.1	Monolingual documents . . . . .	4
1.2	Multilingual documents . . . . .	6
1.3	Mostly monolingual documents . . . . .	7
1.4	Modifiers . . . . .	8
1.5	Troubleshooting . . . . .	8
1.6	Plain . . . . .	9
1.7	Basic language selectors . . . . .	9
1.8	Auxiliary language selectors . . . . .	10
1.9	More on selection . . . . .	10
1.10	Shorthands . . . . .	12
1.11	Package options . . . . .	15
1.12	The base option . . . . .	17
1.13	ini files . . . . .	18
1.14	Selecting fonts . . . . .	25
1.15	Modifying a language . . . . .	27
1.16	Creating a language . . . . .	28
1.17	Digits and counters . . . . .	31
1.18	Accessing language info . . . . .	32
1.19	Hyphenation and line breaking . . . . .	33
1.20	Selection based on BCP 47 tags . . . . .	36
1.21	Selecting scripts . . . . .	37
1.22	Selecting directions . . . . .	37
1.23	Language attributes . . . . .	41
1.24	Hooks . . . . .	42
1.25	Languages supported by babel with ldf files . . . . .	43
1.26	Unicode character properties in luatex . . . . .	44
1.27	Tweaking some features . . . . .	45
1.28	Tips, workarounds, known issues and notes . . . . .	45
1.29	Current and future work . . . . .	46
1.30	Tentative and experimental code . . . . .	46
<b>2</b>	<b>Loading languages with language.dat</b>	<b>47</b>
2.1	Format . . . . .	47
<b>3</b>	<b>The interface between the core of babel and the language definition files</b>	<b>48</b>
3.1	Guidelines for contributed languages . . . . .	49
3.2	Basic macros . . . . .	50
3.3	Skeleton . . . . .	51
3.4	Support for active characters . . . . .	52
3.5	Support for saving macro definitions . . . . .	52
3.6	Support for extending macros . . . . .	52
3.7	Macros common to a number of languages . . . . .	53
3.8	Encoding-dependent strings . . . . .	53
<b>4</b>	<b>Changes</b>	<b>57</b>
4.1	Changes in babel version 3.9 . . . . .	57
<b>II</b>	<b>Source code</b>	<b>57</b>

<b>5</b>	<b>Identification and loading of required files</b>	<b>58</b>
<b>6</b>	<b>locale directory</b>	<b>58</b>
<b>7</b>	<b>Tools</b>	<b>58</b>
7.1	Multiple languages . . . . .	62
7.2	The Package File ( <code>\LaTeX</code> , <code>babel.sty</code> ) . . . . .	63
7.3	<code>base</code> . . . . .	65
7.4	<code>key=value</code> options and other general option . . . . .	67
7.5	Conditional loading of shorthands . . . . .	68
7.6	Cross referencing macros . . . . .	69
7.7	Marks . . . . .	72
7.8	Preventing clashes with other packages . . . . .	73
7.8.1	<code>ifthen</code> . . . . .	73
7.8.2	<code>varioref</code> . . . . .	74
7.8.3	<code>hhline</code> . . . . .	74
7.8.4	<code>hyperref</code> . . . . .	75
7.8.5	<code>fancyhdr</code> . . . . .	75
7.9	Encoding and fonts . . . . .	75
7.10	Basic bidi support . . . . .	77
7.11	Local Language Configuration . . . . .	80
<b>8</b>	<b>The kernel of Babel (<code>babel.def</code>, <code>common</code>)</b>	<b>84</b>
8.1	Tools . . . . .	84
<b>9</b>	<b>Multiple languages</b>	<b>85</b>
9.1	Selecting the language . . . . .	87
9.2	Errors . . . . .	96
9.3	Hooks . . . . .	98
9.4	Setting up language files . . . . .	100
9.5	Shorthands . . . . .	102
9.6	Language attributes . . . . .	112
9.7	Support for saving macro definitions . . . . .	114
9.8	Short tags . . . . .	115
9.9	Hyphens . . . . .	115
9.10	Multiencoding strings . . . . .	117
9.11	Macros common to a number of languages . . . . .	123
9.12	Making glyphs available . . . . .	123
9.12.1	Quotation marks . . . . .	123
9.12.2	Letters . . . . .	125
9.12.3	Shorthands for quotation marks . . . . .	125
9.12.4	Umlauts and tremas . . . . .	126
9.13	Layout . . . . .	128
9.14	Load engine specific macros . . . . .	128
9.15	Creating and modifying languages . . . . .	129
<b>10</b>	<b>Adjusting the Babel bahavior</b>	<b>144</b>
<b>11</b>	<b>Loading hyphenation patterns</b>	<b>145</b>
<b>12</b>	<b>Font handling with <code>fontspec</code></b>	<b>150</b>

<b>13</b>	<b>Hooks for XeTeX and LuaTeX</b>	<b>155</b>
13.1	XeTeX . . . . .	155
13.2	Layout . . . . .	157
13.3	LuaTeX . . . . .	158
13.4	Southeast Asian scripts . . . . .	164
13.5	CJK line breaking . . . . .	168
13.6	Automatic fonts and ids switching . . . . .	168
13.7	Layout . . . . .	176
13.8	Auto bidi with basic and basic-r . . . . .	178
<b>14</b>	<b>Data for CJK</b>	<b>189</b>
<b>15</b>	<b>The ‘nil’ language</b>	<b>189</b>
<b>16</b>	<b>Support for Plain T<sub>E</sub>X (plain.def)</b>	<b>190</b>
16.1	Not renaming hyphen.tex . . . . .	190
16.2	Emulating some L <sup>A</sup> T <sub>E</sub> X features . . . . .	191
16.3	General tools . . . . .	191
16.4	Encoding related macros . . . . .	195
<b>17</b>	<b>Acknowledgements</b>	<b>198</b>

## Troubleshooting

Paragraph ended before \UTFviii@three@octets was complete . . . . .	5
No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the format . . . . .	5
You are loading directly a language style . . . . .	8
Unknown language ‘LANG’ . . . . .	8
Argument of \language@active@arg” has an extra } . . . . .	12
Package fontspec Warning: ‘Language ‘LANG’ not available for font ‘FONT’ with script ‘SCRIPT’ ‘Default’ language used instead’ . . . . .	27
Package babel Info: The following fonts are not babel standard families . . . . .	27

## Part I

# User guide

- This user guide focuses on internationalization and localization with  $\LaTeX$ . 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**, and there are some notes for the latest versions in the babel wiki. The most recent features could be still unstable. Please, report any issues you find in GitHub, which is better than just complaining on an e-mail list or a web forum.
- If you are interested in the  $\TeX$  multilingual support, please join the kadingira mail list. You can follow the development of babel in GitHub (which provides many sample files, too). If you are the author of a package, feel free to send to me a few test files which I'll add to mine, so that possible issues could be caught in the development phase.
- 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 does *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. Many languages are compatible with xetex and luatex. With them you can use babel to localize the documents. When these engines are used, the Latin script is covered by default in current  $\LaTeX$  (provided the document encoding is UTF-8), because the font loader is preloaded and the font is switched to `lmroman`. Other scripts require loading `fontspec`. You may want to set the font attributes with `fontspec`, too.

**EXAMPLE** Here is a simple full example for “traditional”  $\TeX$  engines (see below for xetex and luatex). The packages `fontenc` and `inputenc` do not belong to babel, but they are included in the example because typically you will need them (however, the package `inputenc` may be omitted with  $\LaTeX \geq 2018-04-01$  if the encoding is UTF-8):

PDFTEX

```
\documentclass{article}

\usepackage[T1]{fontenc}
% \usepackage[utf8]{inputenc} % Uncomment if LaTeX < 2018-04-01

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

**EXAMPLE** And now a simple monolingual document in Russian (text from the Wikipedia) with xetex or luatex. Note neither fontenc nor inputenc are necessary, but the document should be encoded in UTF-8 and a so-called Unicode font must be loaded (in this example \babelfont is used, described below).

LUATEX/XETEX

```
\documentclass{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}
```

**TROUBLESHOOTING** A common source of trouble is a wrong setting of the input encoding. Depending on the  $\text{\LaTeX}$  version you could get the following somewhat cryptic error:

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

Or the more explanatory:

```
! Package inputenc Error: Invalid UTF-8 byte ...
```

Make sure you set the encoding actually used by your editor.

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

**EXAMPLE** In 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, but it is discouraged except if there a real reason to do so:

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

Examples of cases where main is useful are the following.

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

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

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

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

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

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

**EXAMPLE** A full bilingual document follows. The main language is french, which is activated when the document begins. The package inputenc may be omitted with L<sup>A</sup>T<sub>E</sub>X ≥ 2018-04-01 if the encoding is UTF-8.

PDFTEX

```
\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}
```

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

LUATEX/XETEX

```
\documentclass{article}

\usepackage[vietnamese,danish]{babel}

\begin{document}

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

\selectlanguage{vietnamese}

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

\end{document}
```

### 1.3 Mostly monolingual documents

**New 3.39** Very often, multilingual documents consist of a main language with small pieces of text in another languages (words, idioms, short sentences). Typically, all you need is to set the line breaking rules and, perhaps, the font. In such a case, babel now does not require declaring these secondary languages explicitly, because the basic settings are loaded on the fly when the language is selected (and also when provided in the optional argument of \babel font, if used.)

This is particularly useful, too, when there are short texts of this kind coming from an external source whose contents are not known on beforehand (for example, titles in a bibliography). At this regard, it is worth remembering that \babel font does not load any font until required, so that it can be used just in case.

**EXAMPLE** A trivial document is:



```

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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}

```

## 1.4 Modifiers

**New 3.9c** The basic behavior of some languages can be modified when loading babel by means of *modifiers*. They are set after the language name, and are prefixed with a dot (only when the language is set as package option – neither global options nor the main key accepts them). An example is (spaces are not significant and they can be added or removed):<sup>1</sup>

```
\usepackage[latin.medieval, spanish.notilde.lcroman, danish]{babel}
```

Attributes (described below) are considered modifiers, ie, you can set an attribute by including it in the list of modifiers. However, modifiers are a more general mechanism.

## 1.5 Troubleshooting

- Loading directly sty files in L<sup>A</sup>T<sub>E</sub>X (ie, `\usepackage{<language>}`) is deprecated and you will get the error:<sup>2</sup>

```

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

```

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

```

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

```

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

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

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

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

## 1.6 Plain

In 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.

**New 3.43** However, if the macro name does not match any language, it will get expanded as expected.

**WARNING** If used inside braces there might be some non-local changes, as this would be roughly equivalent to:

```
{\selectlanguage{<inner-language>} ...}\selectlanguage{<outer-language>}
```

If you want a change which is really local, you must enclose this code with an additional grouping level.

**\foreignlanguage** `{\langle language \rangle}{\langle text \rangle}`

The command `\foreignlanguage` takes two arguments; the second argument is a phrase to be typeset according to the rules of the language named in its first one. This command (1) only switches the extra definitions and the hyphenation rules for the language, *not* the names and dates, (2) does not send information about the language to auxiliary files (i.e.,

---

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

the surrounding language is still in force), and (3) it works even if the language has not been set as package option (but in such a case it only sets the hyphenation patterns and a warning is shown). With the `bidi` option, it also enters in horizontal mode (this is not done always for backwards compatibility).

## 1.8 Auxiliary language selectors

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

The environment `otherlanguage` does basically the same as `\selectlanguage`, except that language change is (mostly) local to the environment.

Actually, there might be some non-local changes, as this environment is roughly equivalent to:

```
\begingroup
\selectlanguage{<inner-language>}
...
\endgroup
\selectlanguage{<outer-language>}
```

If you want a change which is really local, you must enclose this environment with an additional grouping, like braces `{}`.

Spaces after the environment are ignored.

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

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

This environment was originally intended for intermixing left-to-right typesetting with right-to-left typesetting in engines not supporting a change in the writing direction inside a line. However, by default it never complied with the documented 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}`  $\langle\text{language}\rangle$  ... `\end{hyphenrules}`

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

## 1.9 More on selection

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

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

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

**EXAMPLE** With

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

you can write

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

and

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

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

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

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

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

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

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

```
\babelensure{polish}
```

By default only the basic captions and `\today` are redefined, but you can add further macros with the key `include` in the optional argument (without commas). Macros not to be modified are listed in `exclude`. You can also enforce a font encoding with `fontenc`.<sup>5</sup> A couple of examples:

```
\babelensure[include=\Today]{spanish}
\babelensure[fontenc=T5]{vietnamese}
```

They are activated when the language is selected (at the `afterextras` event), and it makes some assumptions which could not be fulfilled in some languages. Note also you should include only macros defined by the language, not global macros (eg,  $\TeX$  of `\dag`). With `ini` files (see below), captions are ensured by default.

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

## 1.10 Shorthands

A *shorthand* is a sequence of one or two characters that expands to arbitrary TeX code. Shorthands can be used for different kinds of things, for example: (1) in some languages shorthands such as "a are defined to be able to hyphenate the word if the encoding is OT1; (2) in some languages shorthands such as ! are used to insert the right amount of white space; (3) several kinds of discretionaries and breaks can be inserted easily with "-", "=", etc. The package inputenc as well as xetex and luatex have alleviated entering non-ASCII characters, but minority languages and some kinds of text can still require characters not directly available on the keyboards (and sometimes not even as separated or precomposed Unicode characters). As to the point 2, now pdfTeX provides \knbccode, 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).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

However, the behavior of hyphens is language-dependent. For example, in languages like Polish and Portuguese, a hard hyphen inside compound words is 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.

`\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

<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.

```
\addto\extrasenglish{\languageshorthands{ngerman}}
```

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

**EXAMPLE** Very often, this is a more convenient way to deactivate shorthands than `\shorthandoff`, for example if you want to define a macro to easy typing phonetic characters with tipa:

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

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

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

**EXAMPLE** Since by default shorthands are not activated until `\begin{document}`, you may use this macro when defining the `\title` in the preamble:

```
\title{Documento científico\babelshorthand{"-}técnico}
```

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

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

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

**Basque** " ' ~

**Breton** : ; ? !

**Catalan** " ' `

**Czech** " -

**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>

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

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

**\ifbabelshorthand**  $\{\langle character \rangle\}\{\langle true \rangle\}\{\langle false \rangle\}$

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

**\aliasshorthand**  $\{\langle original \rangle\}\{\langle alias \rangle\}$

The command `\aliasshorthand` can be used to let another character perform the same functions as the default shorthand character. If one prefers for example to use the character `/` over `"` in typing Polish texts, this can be achieved by entering `\aliasshorthand{"}{/}`. For the reasons in the warning below, usage of this macro is not recommended.

**NOTE** The substitute character must *not* have been declared before as shorthand (in such a case, `\aliasshorthands` is ignored).

**EXAMPLE** The following example shows how to replace a shorthand by another

```
\aliasshorthand{~}{^}  
\AtBeginDocument{\shorthandoff{~}}
```

**WARNING** Shorthands remember somehow the original character, and the fallback value is that of the latter. So, in this example, if no shorthand is found, `^` expands to a non-breaking space, because this is the value of `~` (internally, `^` still calls `\active@char~` or `\normal@char~`). Furthermore, if you change the system value of `^` with `\defineshorthand` nothing happens.

## 1.11 Package options

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

- KeepShorthandsActive** Tells babel not to deactivate shorthands after loading a language file, so that they are also available in the preamble.
- activeacute** For some languages babel supports this options to set `'` as a shorthand in case it is not done by default.
- activegrave** Same for ```.
- shorthands=**  $\langle char \rangle \langle char \rangle \dots \mid \text{off}$   
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  $\LaTeX$  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.



<b>safe=</b>	none   ref   bib
	Some $\LaTeX$ macros are redefined so that using shorthands is safe. With <code>safe=bib</code> only <code>\nocite</code> , <code>\bibcite</code> and <code>\bibitem</code> are redefined. With <code>safe=ref</code> only <code>\newlabel</code> , <code>\ref</code> and <code>\pageref</code> are redefined (as well as a few macros from <code>varioref</code> and <code>ifthen</code> ). With <code>safe=none</code> no macro is redefined. This option is strongly recommended, because a good deal of incompatibilities and errors are related to these redefinitions. As of <b>New 3.34</b> , in $\epsilon\TeX$ based engines (ie, almost every engine except the oldest ones) shorthands can be used in these macros (formerly you could not).
<b>math=</b>	active   normal
	Shorthands are mainly intended for text, not for math. By setting this option with the value <code>normal</code> they are deactivated in math mode (default is <code>active</code> ) and things like <code>#{a'}</code> (a closing brace after a shorthand) are not a source of trouble anymore.
<b>config=</b>	$\langle file \rangle$
	Load $\langle file \rangle$ .cfg instead of the default config file <code>bblopts.cfg</code> (the file is loaded even with <code>noconfigs</code> ).
<b>main=</b>	$\langle language \rangle$
	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.
<b>headfoot=</b>	$\langle language \rangle$
	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.
<b>noconfigs</b>	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.
<b>showlanguages</b>	Prints to the log the list of languages loaded when the format was created: number (remember dialects can share it), name, hyphenation file and exceptions file.
<b>nocase</b>	<b>New 3.91</b> Language settings for uppercase and lowercase mapping (as set by <code>\SetCase</code> ) are ignored. Use only if there are incompatibilities with other packages.
<b>silent</b>	<b>New 3.91</b> No warnings and no <i>infos</i> are written to the log file. <sup>9</sup>
<b>strings=</b>	generic   unicode   encoded   $\langle label \rangle$   $\langle font encoding \rangle$
	Selects the encoding of strings in languages supporting this feature. Predefined labels are <code>generic</code> (for traditional $\TeX$ , LICR and ASCII strings), <code>unicode</code> (for engines like <code>xetex</code> and <code>luatex</code> ) and <code>encoded</code> (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 <code>\MakeUpper</code> case and the like (this feature misuses some internal $\LaTeX$ tools, so use it only as a last resort).
<b>hyphenmap=</b>	off   first   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=** default | basic | basic-r | bidi-l | bidi-r

**New 3.14** Selects the bidi algorithm to be used in `luatex` and `xetex`. See sec. 1.22.

**layout=**

**New 3.16** Selects which layout elements are adapted in bidi documents. See sec. 1.22.

## 1.12 The base option

With this package option `babel` just loads some basic macros (those in `switch.def`), defines `\AfterBabelLanguage` and exits. It also selects the hyphenation patterns for the last language passed as option (by its name in `language.dat`). There are two main uses: classes and packages, and as a last resort in case there are, for some reason, incompatible languages. It can be used if you just want to select the hyphenation patterns of a single language, too.

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

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

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

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

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

---

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

```

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

```

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

### 1.13 ini files

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

ini files are not meant only for babel, and they have been devised as a resource for other packages. To ease interoperability between T<sub>E</sub>X and other systems, they are identified with the BCP 47 codes as preferred by the Unicode Common Language Data Repository, which was used as source for most of the data provided by these files, too (the main exception being the \dotsname strings).

Most of them set the date, and many also the captions (Unicode and LICR). They will be evolving with the time to add more features (something to keep in mind if backward compatibility is important). The following section shows how to make use of them currently (by means of \babelprovide), but a higher interface, based on package options, is under study. In other words, \babelprovide is mainly meant 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.

LUATEX/XETEX

```

\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}

```

**NOTE** The ini files just define and set some parameters, but the corresponding behavior is not always implemented. Also, there are some limitations in the engines. A few remarks follow:

**Arabic** Monolingual documents mostly work in luatex, but it must be fine tuned, and a recent version of fontspec/loaotfloat is required. In xetex babel resorts to the bidi package, which seems to work.

**Hebrew** Niqud marks seem to work in both engines, but cantillation marks are misplaced (xetex or luatex with Harfbuzz seems better, but still problematic).

**Devanagari** In luatex and the the default renderer many fonts work, but some others do not, the main issue being the ‘ra’. It is advisable to set explicitly the script to either deva or dev2, eg:

`\newfontscript{Devanagari}{deva}`

Other Indic scripts are still under development in the default `luatex` renderer, but should work with the option `Renderer=Harfbuzz` in `FONTSPEC`. They also work with `xetex`, although fine tuning the font behaviour is not always possible.

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

```
\babelprovide[import,hyphenrules=+]{lao}
\babelpatterns[lao]{໐໑ ໑໒ ໑໓ ໑໔ ໑໕} % Random
```

**East Asia scripts** Settings for either Simplified or Traditional should work out of the box, with basic line breaking. Although for a few words and short texts the ini files should be fine, CJK texts are best set with a dedicated framework (CJK, luatexja, kotex, CTeX, etc.). This is what the class `ltjbook` does with `luatex`, which can be used in conjunction with the `ldf` for `japanese`, because the following piece of code loads `luatexja`:

```
\documentclass{ltjbook}
\usepackage[japanese]{babel}
```

**NOTE** Wikipedia defines a *locale* as follows: “In computing, a locale is a set of parameters that defines the user’s language, region and any special variant preferences that the user wants to see in their user interface. Usually a locale identifier consists of at least a language code and a country/region code.” Babel is moving gradually from the old and fuzzy concept of *language* to the more modern of *locale*. Note each locale is by itself a separate “language”, which explains why there are so many files. This is on purpose, so that possible variants can be created and/or redefined easily.

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

af	Afrikaans <sup>ul</sup>	az-Latn	Azerbaijani
agg	Aghem	az	Azerbaijani <sup>ul</sup>
ak	Akan	bas	Basaa
am	Amharic <sup>ul</sup>	be	Belarusian <sup>ul</sup>
ar	Arabic <sup>ul</sup>	bem	Bemba
ar-DZ	Arabic <sup>ul</sup>	bez	Bena
ar-MA	Arabic <sup>ul</sup>	bg	Bulgarian <sup>ul</sup>
ar-SY	Arabic <sup>ul</sup>	bm	Bambara
as	Assamese	bn	Bangla <sup>ul</sup>
asa	Asu	bo	Tibetan <sup>u</sup>
ast	Asturian <sup>ul</sup>	brx	Bodo
az-Cyrl	Azerbaijani	bs-Cyrl	Bosnian

bs-Latn	Bosnian <sup>ul</sup>	gu	Gujarati
bs	Bosnian <sup>ul</sup>	guz	Gusii
ca	Catalan <sup>ul</sup>	gv	Manx
ce	Chechen	ha-GH	Hausa
cgg	Chiga	ha-NE	Hausa <sup>l</sup>
chr	Cherokee	ha	Hausa
ckb	Central Kurdish	haw	Hawaiian
cop	Coptic	he	Hebrew <sup>ul</sup>
cs	Czech <sup>ul</sup>	hi	Hindi <sup>u</sup>
cu	Church Slavic	hr	Croatian <sup>ul</sup>
cu-Cyrs	Church Slavic	hsb	Upper Sorbian <sup>ul</sup>
cu-Glag	Church Slavic	hu	Hungarian <sup>ul</sup>
cy	Welsh <sup>ul</sup>	hy	Armenian <sup>u</sup>
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

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

yi	Yiddish	zh-Hans-SG	Chinese
yo	Yoruba	zh-Hans	Chinese
yue	Cantonese	zh-Hant-HK	Chinese
zgh	Standard Moroccan Tamazight	zh-Hant-MO	Chinese
		zh-Hant	Chinese
zh-Hans-HK	Chinese	zh	Chinese
zh-Hans-MO	Chinese	zu	Zulu

---

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

---

aghem	brazilian
akan	breton
albanian	british
american	bulgarian
amharic	burmese
arabic	canadian
arabic-algeria	cantonese
arabic-DZ	catalan
arabic-morocco	centralatlastamazight
arabic-MA	centralkurdish
arabic-syria	chechen
arabic-SY	cherokee
armenian	chiga
assamese	chinese-hans-hk
asturian	chinese-hans-mo
asu	chinese-hans-sg
australian	chinese-hans
austrian	chinese-hant-hk
azerbaijani-cyrillic	chinese-hant-mo
azerbaijani-cyrl	chinese-hant
azerbaijani-latin	chinese-simplified-hongkongsarchina
azerbaijani-latn	chinese-simplified-macausarchina
azerbaijani	chinese-simplified-singapore
bafia	chinese-simplified
bambara	chinese-traditional-hongkongsarchina
basaa	chinese-traditional-macausarchina
basque	chinese-traditional
belarusian	chinese
bemba	churchslavic
bena	churchslavic-cyrs
bengali	churchslavic-oldcyrillic <sup>13</sup>
bodo	churchsslavic-glag
bosnian-cyrillic	churchsslavic-glagolitic
bosnian-cyrl	colognian
bosnian-latin	cornish
bosnian-latn	croatian
bosnian	czech

---

<sup>13</sup>The name in the CLDR is Old Church Slavonic Cyrillic, but it has been shortened for practical reasons.

danish  
duala  
dutch  
dzongkha  
embu  
english-au  
english-australia  
english-ca  
english-canada  
english-gb  
english-newzealand  
english-nz  
english-unitedkingdom  
english-unitedstates  
english-us  
english  
esperanto  
estonian  
ewe  
ewondo  
faroeese  
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  
northern sami  
northndebele  
norwegianbokmal  
norwegiannynorsk  
nswissgerman  
nuer  
nyankole  
nynorsk  
occitan  
oriya  
oromo  
ossetic  
pashto  
persian  
piedmontese  
polish  
portuguese-br  
portuguese-brazil  
portuguese-portugal  
portuguese-pt  
portuguese  
punjabi-arab  
punjabi-arabic  
punjabi-gurmukhi  
punjabi-guru  
punjabi  
quechua  
romanian  
romansh  
rombo  
rundi  
russian  
rwa  
sakha  
samburu  
samin  
sango  
sangu  
sanskrit-beng  
sanskrit-bengali

sanskrit-deva  
sanskrit-devanagari  
sanskrit-gujarati  
sanskrit-gujr  
sanskrit-kannada  
sanskrit-knda  
sanskrit-malayalam  
sanskrit-mlym  
sanskrit-telu  
sanskrit-telugu  
sanskrit  
scottishgaelic  
sena  
serbian-cyrillic-bosniaherzegovina  
serbian-cyrillic-kosovo  
serbian-cyrillic-montenegro  
serbian-cyrillic  
serbian-cyrl-ba  
serbian-cyrl-me  
serbian-cyrl-xk  
serbian-cyrl  
serbian-latin-bosniaherzegovina  
serbian-latin-kosovo  
serbian-latin-montenegro  
serbian-latin  
serbian-latn-ba  
serbian-latn-me  
serbian-latn-xk  
serbian-latn  
serbian  
shambala  
shona  
sichuanyi  
sinhala  
slovak  
slovene  
slovenian  
soga  
somali  
spanish-mexico  
spanish-mx  
spanish  
standardmoroccantamazight  
swahili  
swedish  
swissgerman  
tachelhit-latin  
tachelhit-latn  
tachelhit-tfng  
tachelhit-tifinagh  
tachelhit  
taita  
tamil  
tasawaq

telugu	uzbek-latin
teso	uzbek-latn
thai	uzbek
tibetan	vai-latin
tigrinya	vai-latn
tongan	vai-vai
turkish	vai-vaii
turkmen	vai
ukenglish	vietnam
ukrainian	vietnamese
upporsorbian	vunjo
urdu	walser
usenglish	welsh
usorbian	westernfrisian
uyghur	yangben
uzbek-arab	yiddish
uzbek-arabic	yoruba
uzbek-cyrillic	zarma
uzbek-cyrl	zulu afrikaans

---

### Modifying and adding values to ini files

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

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

## 1.14 Selecting fonts

**New 3.15** Babel provides a high level interface on top of `fontspec` to select fonts. There is no need to load `fontspec` explicitly – babel does it for you with the first `\babelfont`.<sup>14</sup>

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

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

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

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

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

Babel takes care of the font language and the font script when languages are selected (as well as the writing direction); see the recognized languages above. In most cases, you will

---

<sup>14</sup>See also the package `combofont` for a complementary approach.

not need *font-options*, which is the same as in fontspec, but you may add further key/value pairs if necessary.

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

LUATEX/XETEX

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עברית} svenska.

\end{document}
```

If on the other hand you have to resort to different fonts, you could replace the red line above with, say:

LUATEX/XETEX

```
\babelfont{rm}{Iwona}
\babelfont[hebrew]{rm}{FreeSerif}
```

`\babelfont` can be used to implicitly define a new font family. Just write its name instead of `rm`, `sf` or `tt`. This is the preferred way to select fonts in addition to the three basic families.

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

LUATEX/XETEX

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

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

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

LUATEX/XETEX

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

This makes sure the OpenType script for Devanagari is `deva` and not `dev2`, in case it is not detected correctly. You may also pass some options to fontspec: with `silent`, the warnings about unavailable scripts or languages are not shown (they are only really useful when the document format is being set up).

**NOTE** Directionality is a property affecting margins, indentation, column order, etc., not just text. Therefore, it is under the direct control of the language, which applies both the script and the direction to the text. As a consequence, there is no need to set `Script` when declaring a font with `\babelfont` (nor `Language`). In fact, it is even discouraged.

**NOTE** `\fontspec` is not touched at all, only the preset font families (`rm`, `sf`, `tt`, and the like). If a language is switched when an *ad hoc* font is active, or you select the font with this command, neither the script nor the language is passed. You must add them by hand. This is by design, for several reasons—for example, each font has its own set of features and a generic setting for several of them could be problematic, and also a “lower-level” font selection is useful.

**NOTE** The keys `Language` and `Script` just pass these values to the *font*, and do *not* set the script for the *language* (and therefore the writing direction). In other words, the `ini` file or `\babelprovide` provides default values for `\babelfont` if omitted, but the opposite is not true. See the note above for the reasons of this behavior.

**WARNING** Using `\setxxxxfont` and `\babelfont` at the same time is discouraged, but very often works as expected. However, be aware with `\setxxxxfont` the language system will not be set by `babel` and should be set with `fontspec` if necessary.

**TROUBLESHOOTING** *Package fontspec Warning: ‘Language ‘LANG’ not available for font ‘FONT’ with script ‘SCRIPT’ ‘Default’ language used instead’.*

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

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

**This is *not* and error.** `babel` assumes that if you are using `\babelfont` for a family, very likely you want to define the rest of them. If you don’t, you can find some inconsistencies between families. This checking is done at the beginning of the document, at a point where we cannot know which families will be used.

Actually, there is no real need to use `\babelfont` in a monolingual document, if you set the language system in `\setmainfont` (or not, depending on what you want).

As the message explains, *there is nothing intrinsically wrong* with not defining all the families. In fact, there is nothing intrinsically wrong with not using `\babelfont` at all. But you must be aware that this may lead to some problems.

## 1.15 Modifying a language

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

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

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

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

- 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** Do *not* redefine a caption in the following way:

```
\AtBeginDocument{\renewcommand\contentsname{Foo}}
```

The changes may be discarded with a language selector, and the original value restored.

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

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

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

first loads `danish.ldf`, and then redefines the captions for danish (as provided by the `ini` file) and prevents hyphenation. The rest of the language definitions are not touched.

## 1.16 Creating a language

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

`\babelprovide` [`<options>`] {`<language-name>`}

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

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

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

```
Package babel Warning: \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}
```

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

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

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

The main language is not changed (danish in this example). So, you must add `\selectlanguage{arhinish}` or other selectors where necessary. If the language has been loaded as an argument in `\documentclass` or `\usepackage`, then `\babelprovide` redefines the requested data.

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

**New 3.13** Imports data from an ini file, including captions, 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. So, the previous example could be written:

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

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 (french, breton, and occitan).

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

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

Loads only the strings. For example:

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

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

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

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

If none of the listed hyphenrules exist, the default 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=**  $\langle$ *script-name* $\rangle$

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

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

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

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

**onchar=** ids | fonts

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

**mapfont=** direction

Assigns the font for the writing direction of this language (only with `bidi=basic`). Whenever possible, instead of this option use `onchar`, based on the script, which usually makes more sense. More precisely, what `mapfont=direction` means is, ‘when a character has the same direction as the script for the “provided” language, then change its font to that set for this language’. There are 3 directions, following the bidi Unicode algorithm, namely, Arabic-like, Hebrew-like and left to right. So, there should be at most 3 directives of this kind.

**intraspace=**  $\langle base \rangle \langle shrink \rangle \langle stretch \rangle$

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

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

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

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

## 1.17 Digits and counters

**New 3.20** About thirty ini files define a field named `digits.native`. When it is present, two macros are created: `\<language>digits` and `\<language>counter` (only xetex and luatex). With the first, a string of ‘Latin’ digits are converted to the native digits of that language; the second takes a counter name as argument. With the option `maparabic` in `\babelprovide`, `\arabic` is redefined to produce the native digits (this is done *globally*, to avoid inconsistencies in, for example, page numbering, and note as well dates do not rely on `\arabic`.)

For example:

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

Languages providing native digits in all or some variants are:

Arabic	Persian	Lao	Odia	Urdu
Assamese	Gujarati	Northern Luri	Punjabi	Uzbek
Bangla	Hindi	Malayalam	Pashto	Vai
Tibetar	Khmer	Marathi	Tamil	Cantonese
Bodo	Kannada	Burmese	Telugu	Chinese
Central Kurdish	Konkani	Mazanderani	Thai	
Dzongkha	Kashmiri	Nepali	Uyghur	



**New 3.30** With luatex there is an alternative approach for mapping digits, namely, `mapdigits`. Conversion is based on the language and it is applied to the typeset text (not math, PDF bookmarks, etc.) before bidi and fonts are processed (ie, to the node list as generated by the T<sub>E</sub>X code). This means the local digits have the correct bidirectional behavior (unlike `Numbers=Arabic` in fontspec, which is not recommended).

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

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

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

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

The styles are:

**Ancient Greek** `lower.ancient`, `upper.ancient`  
**Arabic** `abjad`, `maghrebi.abjad`  
**Belarusan, Bulgarian, Macedonian, Serbian** `lower`, `upper`  
**Hebrew** `letters` (neither `geresh` nor `gershayim yet`)  
**Hindi** `alphabetic`  
**Armenian** `lower.letter`, `upper.letter`  
**Japanese** `hiragana`, `hiragana.iroha`, `katakana`, `katakana.iroha`, `circled.katakana`,  
`informal`, `formal`, `cjk-earthly-branch`, `cjk-heavenly-stem`,  
`fullwidth.lower.alpha`, `fullwidth.upper.alpha`  
**Georgian** `letters`  
**Greek** `lower.modern`, `upper.modern`, `lower.ancient`, `upper.ancient` (all with `keraia`)  
**Khmer** `consonant`  
**Korean** `consonant`, `syllabe`, `hanja.informal`, `hanja.formal`, `hangul.formal`,  
`cjk-earthly-branch`, `cjk-heavenly-stem`, `fullwidth.lower.alpha`,  
`fullwidth.upper.alpha`  
**Persian** `abjad`, `alphabetic`  
**Russian** `lower`, `lower.full`, `upper`, `upper.full`  
**Tamil** `ancient`  
**Thai** `alphabetic`  
**Ukrainian** `lower`, `lower.full`, `upper`, `upper.full`  
**Chinese** `cjk-earthly-branch`, `cjk-heavenly-stem`, `fullwidth.lower.alpha`,  
`fullwidth.upper.alpha`

## 1.18 Accessing language info

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

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

`\iflanguage`  $\{\langle language \rangle\}\{\langle true \rangle\}\{\langle false \rangle\}$

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  $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ sense, as a set of hyphenation patterns, and *not* as its babel name. This macro takes three arguments. The first argument is the name of a language; the second and third arguments are the actions to take if the result of the test is true or false respectively.

`\localeinfo`  $\{\langle field \rangle\}$

**New 3.38** If an ini file has been loaded for the current language, you may access the information stored in it. This macros is fully expandable and the available fields are:

`name.english` as provided by the Unicode CLDR.

`tag.ini` is the tag of the ini file (the way this file is identified in its name).

`tag.bcp47` is the BCP 47 language tag.

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

`script.name` as provided by the Unicode CLDR.

`script.tag.bcp47` is the BCP 47 language tag of the script used by this locale.

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

`\getlocaleproperty`  $\{\langle macro \rangle\}\{\langle locale \rangle\}\{\langle property \rangle\}$

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

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

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

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

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

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

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

## 1.19 Hyphenation and line breaking

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

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

`\babelhyphen`  $\ast\{\langle text \rangle\}$

**New 3.9a** It is customary to classify hyphens in two types: (1) *explicit* or *hard hyphens*, which in  $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  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  $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  terms, a “discretionary”; a *hard hyphen* is a hyphen with a breaking opportunity after it. A further type is a *non-breaking hyphen*, a hyphen without a breaking opportunity.

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

- `\babelhyphen{soft}` and `\babelhyphen{hard}` are self explanatory.
- `\babelhyphen{repeat}` inserts a hard hyphen which is repeated at the beginning of the next line, as done in languages like Polish, Portuguese and Spanish.
- `\babelhyphen{nobreak}` inserts a hard hyphen without a break after it (even if a space follows).
- `\babelhyphen{empty}` inserts a break opportunity without a hyphen at all.
- `\babelhyphen{<text>}` is a hard “hyphen” using `<text>` instead. A typical case is `\babelhyphen{/}`.

With all of them, hyphenation in the rest of the word is enabled. If you don’t want to enable it, there is a starred counterpart: `\babelhyphen*{soft}` (which in most cases is equivalent to the original \-), `\babelhyphen*{hard}`, etc.

Note `hard` is also good for isolated prefixes (eg, *anti-*) and `nobreak` for isolated suffixes (eg, *-ism*), but in both cases `\babelhyphen*{nobreak}` is usually better.

There are also some differences with  $\TeX$ : (1) the character used is that set for the current font, while in  $\TeX$  it is hardwired to - (a typical value); (2) the hyphen to be used in fonts with a negative `\hyphenchar` is -, like in  $\TeX$ , 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.

**NOTE** Using `\babelhyphenation` with Southeast Asian scripts is mostly pointless. But with `\babelpatterns` (below) you may fine-tune line breaking (only `luatex`). Even if there are no patterns for the language, you can add at least some typical cases.

**\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.

**New 3.31** (Only *luatex*.) With `\babelprovide` and imported CJK languages, a simple generic line breaking algorithm (push-out-first) is applied, based on a selection of the Unicode rules ( **New 3.32** it is disabled in verbatim mode, or more precisely when the `hyphenrules` are set to `nohyphenation`). It can be activated alternatively by setting explicitly the `intraspace`.

**New 3.27** Interword spacing for Thai, Lao and Khemer is activated automatically if a language with one of those scripts are loaded with `\babelprovide`. See the sample on the babel repository. With both Unicode engines, spacing is based on the “current” em unit (the size of the previous char in *luatex*, and the font size set by the last `\selectfont` in *xetex*).

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

**New 3.37-3.39** With *luatex* it is now possible to define non-standard hyphenation rules, like `f-f`  $\rightarrow$  `ff-f`, repeated hyphens, ranked ruled (or more precisely, ‘penalized’ hyphenation points), and so on. No rules are currently provided by default, but they can be defined as shown in the following example, where `{1}` is the first captured char (between `()` in the pattern):

```
\babelposthyphenation{german}{([fmtrp]) | {1}}
{
  { no = {1}, pre = {1}{1}- }, % Replace first char with disc
  remove,                    % Remove automatic disc (2nd node)
  {}                          % Keep last char, untouched
}
```

In the replacements, a captured char may be mapped to another, too. For example, if the first capture reads `([íú])`, the replacement could be `{1|íú|íú}`, which maps `í` to `í`, and `ú` to `ú`, so that the diaeresis is removed.

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

See the babel wiki for a more detailed description and some examples. It also describes an additional replacement type with the key `string`.

**EXAMPLE** Although the main purpose of this command is non-standard hyphenation, it may actually be used for other transformations (after hyphenation is applied, so you must take discretionaries into account). For example, you can use the `string` replacement to replace a character (or series of them) by another character (or series of them). Thus, to enter `ž` as `zh` and `š` as `sh` in a newly created locale for transliterated Russian:

<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.

```

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

```

In other words, it is a quite general tool. (A counterpart `\babelprehyphenation` is on the way.)

## 1.20 Selection based on BCP 47 tags

**New 3.43** The recommended way to select languages is that described at the beginning of this document. However, BCP 47 tags are becoming customary, particularly in document (or parts of documents) generated by external sources, and therefore babel will provide a set of tools to select the locales in different situations, adapted to the particular needs of each case. Currently, babel provides autoloading of locales as described in this section. In these contexts autoloading is particularly important because we may not know on beforehand which languages will be requested.

It must be activated explicitly, because it is primarily meant for special tasks. Mapping from BCP 47 codes to locale names are not hardcoded in babel. Instead the data is taken from the ini files, which means currently about 250 tags are already recognized. Babel performs a simple lookup in the following way: `fr-Latn-FR` → `fr-Latn` → `fr-FR` → `fr`. Languages with the same resolved name are considered the same. Case is normalized before, so that `fr-latn-fr` → `fr-Latn-FR`. If a tag and a name overlap, the tag takes precedence.

Here is a minimal example:

```

\documentclass{article}

\usepackage[danish]{babel}

\babeladjust{ autload.bcp47 = on }

\begin{document}

\today

\selectlanguage{fr-CA}

\today

\end{document}

```

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

The behaviour is adjusted with `\babeladjust` with the following parameters:

`autload.bcp47` with values on and off.

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

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

## 1.21 Selecting scripts

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

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

`\ensureascii`  $\langle text \rangle$

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

If non-ASCII encodings are not loaded (or no encoding at all), it is no-op (also `\TeX` and `\LaTeX` are not redefined); otherwise, `\ensureascii` switches to the encoding at the beginning of the document if ASCII-savvy, or else the last ASCII-savvy encoding loaded. For example, if you load LY1, LGR, then it is set to LY1, but if you load LY1, T2A it is set to T2A. The symbol encodings TS1, T3, and TS3 are not taken into account, since they are not used for “ordinary” text (they are stored in `\BabelNonText`, used in some special cases when no Latin encoding is explicitly set).

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

## 1.22 Selecting directions

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

**WARNING** 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. This applies to text; there is a basic support for **graphical** elements, including the picture environment (with `pict2e`) and `pfg/tikz`. Also, indexes and the like are under study, as well as math (there is progress in the latter, too, but for example cases may fail).

<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.

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

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

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

```

\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

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

\end{document}

```

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

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

```

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

```

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

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

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

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

**counters** required in all engines (except `luatex` with `bidi=basic`) to reorder section numbers and the like (eg, `\subsection`.`\section`); required in `xetex` and `pdftex` for counters in general, as well as in `luatex` with `bidi=default`; required in `luatex` for numeric footnote marks `>9` with `bidi=basic-r` (but *not* with `bidi=basic`); note, however, it 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>18</sup>

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**New 3.17** Something like:

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

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

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

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

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

## 1.23 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).

## 1.24 Hooks

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

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

The same name can be applied to several events. Hooks may be enabled and disabled for all defined events with `\EnableBabelHook{<name>}`, `\DisableBabelHook{<name>}`. Names containing the string `babel` are reserved (they are used, for example, by `\usesshortands*` to add a hook for the event `afterextras`). **New 3.33** They may be also applied to a specific language with the optional argument; language-specific settings are executed after global ones.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## 1.25 Languages supported by babel with ldf files

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

**Afrikaans** afrikaans

**Azerbaijani** azerbaijani

**Basque** basque

**Breton** breton

**Bulgarian** bulgarian

**Catalan** catalan

**Croatian** croatian

**Czech** czech

**Danish** danish

**Dutch** dutch

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

**Esperanto** esperanto

**Estonian** estonian

**Finnish** finnish

**French** french, francais, canadien, acadian

**Galician** galician

**German** austrian, german, germanb, ngerman, naustrian

**Greek** greek, polutonikogreek

**Hebrew** hebrew

**Icelandic** icelandic

**Indonesian** indonesian, bahasa, indon, bahasai

**Interlingua** interlingua

**Irish Gaelic** irish

**Italian** italian

**Latin** latin

**Lower Sorbian** lowersorbian

**Malay** malay, melayu, bahasam

**North Sami** samin

**Norwegian** norsk, nynorsk

**Polish** polish  
**Portuguese** portuguese, portuges<sup>19</sup>, brazilian, brazil  
**Romanian** romanian  
**Russian** russian  
**Scottish Gaelic** scottish  
**Spanish** spanish  
**Slovakian** slovak  
**Slovenian** slovene  
**Swedish** swedish  
**Serbian** serbian  
**Turkish** turkish  
**Ukrainian** ukrainian  
**Upper Sorbian** uppersorbian  
**Welsh** welsh

There are more languages not listed above, including hindi, thai, thaicjk, latvian, turkmen, magyar, mongolian, romansh, lithuanian, spanglish, vietnamese, japanese, pinyin, arabic, farsi, ibygreek, bgreek, serbianc, frenchle, ethiop and friulan. Most of them work out of the box, but some may require extra fonts, encoding files, a preprocessor or even a complete framework (like CJK or luatexja). For example, if you have got the velthuis/devnag package, you can create a file with extension .dn:

```

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

```

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

## 1.26 Unicode character properties in luatex

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

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

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

```

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

```

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

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

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

## 1.27 Tweaking some features

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

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

## 1.28 Tips, workarounds, known issues and notes

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

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

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

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

```
\addto\extrasfrench{\inputencoding{latin1}}
\addto\extrarussian{\inputencoding{koi8-r}}
```

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

- For the hyphenation to work correctly, `lccodes` cannot change, because  $\TeX$  only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.<sup>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  $\TeX$ , not of `babel`. Alternatively, you may use `\usesorthands` to activate `'` and `\defineshortand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).
- `\bibitem` is out of sync with `\selectlanguage` in the `.aux` file. The reason is `\bibitem` uses `\immediate` (and others, in fact), while `\selectlanguage` doesn't. There is no known workaround.

<sup>20</sup>This explains why  $\LaTeX$  assumes the lowercase mapping of T1 and does not provide a tool for multiple mappings. Unfortunately, `\savingshyphcodes` is not a solution either, because `lccodes` for hyphenation are frozen in the format and cannot be changed.

- 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.

## 1.29 Current and future work

The current work is focused on the so-called complex scripts in  $\text{luatex}$ . In 8-bit engines, babel provided a basic support for bidi text as part of the style for Hebrew, but it is somewhat unsatisfactory and internally replaces some hardwired commands by other hardwired commands (generic changes would be much better). Useful additions would be, for example, time, currency, addresses and personal names.<sup>21</sup> But that is the easy part, because they don’t require modifying the  $\text{\TeX}$  internals. Calendars (Arabic, Persian, Indic, etc.) are under study. Also interesting are differences in the sentence structure or related to it. For example, in Basque the number precedes the name (including chapters), in Hungarian “from (1)” is “(1)-ből”, but “from (3)” is “(3)-ból”, in Spanish an item labelled “3.<sup>o</sup>” may be referred to as either “ítem 3.<sup>o</sup>” or “3.<sup>er</sup> ítem”, and so on. An option to manage bidirectional document layout in  $\text{luatex}$  (lists, footnotes, etc.) is almost finished, but xetex required more work. Unfortunately, proper support for xetex requires patching somehow lots of macros and packages (and some issues related to `\specials` remain, like color and hyperlinks), so babel resorts to the bidi package (by Vafa Khalighi). See the babel repository for a small example (xe-bidi).

## 1.30 Tentative and experimental code

See the code section for `\foreignlanguage*` (a new starred version of `\foreignlanguage`).

### Old and deprecated 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:

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

- `\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}
```

## 2 Loading languages with `language.dat`

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

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

### 2.1 Format

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

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

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

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

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

<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.



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

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

A typical error when using `babel` is the following:

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

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

### 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  $\text{\TeX}$  users, so the files have to be coded so that they can be read by both  $\text{\LaTeX}$  and plain  $\text{\TeX}$ . The current format can be checked by looking at the value of the macro `\fmtname`.
- The common part of the `babel` system redefines a number of macros and environments (defined previously in the document style) to put in the names of macros that replace the previously hard-wired texts. These macros have to be defined in the language definition files.
- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are `\<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 you need further assistance and technical advice in the development of language styles, I am willing to help you. And of course, you can make any suggestion you like.

---

<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  $\TeX$  to use a font from the second family when a font from the first family in the given encoding seems to be needed.

### 3.3 Skeleton

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

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

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

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

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

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

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

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

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

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

\EndBabelCommands

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

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

**NOTE** If for some reason you want to load a package in your style, you should be aware it

cannot be done directly in the ldf file, but it can be delayed with `\AtEndOfPackage`. Macros from external packages can be used *inside* definitions in the ldf itself (for example, `\extras<language>`), but if executed directly, the code must be placed inside `\AtEndOfPackage`. A trivial example illustrating these points is:

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

### 3.4 Support for active characters

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

`\initiate@active@char`

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

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

The command `\bbl@activate` is used to change the way an active character expands. `\bbl@activate` ‘switches on’ the active behavior of the character. `\bbl@deactivate` lets the active character expand to its former (mostly) non-active self.

`\declare@shorthand`

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

`\bbl@add@special`  
`\bbl@remove@special`

The  $\TeX$ book states: “Plain  $\TeX$  includes a macro called `\dospecials` that is essentially a set macro, representing the set of all characters that have a special category code.” [4, p. 380] It is used to set text ‘verbatim’. To make this work if more characters get a special category code, you have to add this character to the macro `\dospecial`.  $\TeX$  adds another macro called `\@sanitize` representing the same character set, but without the curly braces. The macros `\bbl@add@special<char>` and `\bbl@remove@special<char>` add and remove the character `<char>` to these two sets.

### 3.5 Support for saving macro definitions

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

`\babel@save`

To save the current meaning of any control sequence, the macro `\babel@save` is provided. It takes one argument, `<csname>`, the control sequence for which the meaning has to be saved.

`\babel@savevariable`

A second macro is provided to save the current value of a variable. In this context, anything that is allowed after the `\the` primitive is considered to be a variable. The macro takes one argument, the `<variable>`.

The effect of the preceding macros is to append a piece of code to the current definition of `\originalTeX`. When `\originalTeX` is expanded, this code restores the previous definition of the control sequence or the previous value of the variable.

### 3.6 Support for extending macros

`\addto`

The macro `\addto{<control sequence>}{<TeX code>}` can be used to extend the definition of

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

a macro. The macro need not be defined (ie, it can be undefined or `\relax`). This macro can, for instance, be used in adding instructions to a macro like `\extrasenglish`. Be careful when using this macro, because depending on the case the assignment could be either global (usually) or local (sometimes). That does not seem very consistent, but this behavior is preserved for backward compatibility. If you are using `etoolbox`, by Philipp Lehman, consider using the tools provided by this package instead of `\addto`.

### 3.7 Macros common to a number of languages

`\bbl@allowhyphens`

In several languages compound words are used. This means that when  $\TeX$  has to hyphenate such a compound word, it only does so at the ‘-’ that is used in such words. To allow hyphenation in the rest of such a compound word, the macro `\bbl@allowhyphens` can be used.

`\allowhyphens`

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

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

`\set@low@box`

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

`\save@sf@q`

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

`\bbl@frenchspacing`

`\bbl@nonfrenchspacing`

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

### 3.8 Encoding-dependent strings

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

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

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

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

`\StartBabelCommands`

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

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

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

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

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

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

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

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

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

\EndBabelCommands
```

A real example is:

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

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

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

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

\StartBabelCommands{german,austrian}{date}
\SetString\monthiiname{Februar}
\SetString\monthiiname{M}{a}rz}
\SetString\monthivname{April}
```

---

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

```

\SetString\monthvname{Mai}
\SetString\monthviname{Juni}
\SetString\monthviiname{Juli}
\SetString\monthviiiname{August}
\SetString\monthixname{September}
\SetString\monthxname{Oktober}
\SetString\monthxiname{November}
\SetString\monthxiiname{Dezenber}
\SetString\today{\number\day.~%
\csname month\romannumeral\month name\endcsname\space
\number\year}

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

\EndBabelCommands

```

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

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

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  $\backslash abday$ ):

```

\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.

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



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

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

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

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

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

**\SetHyphenMap** {*<to-lower-macros>*}

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

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

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

## Part II

# Source code

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

## 5 Identification and loading of required files

*Code documentation is still under revision.*

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

The babel package after unpacking consists of the following files:

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

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

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

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

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

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

## 6 locale directory

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

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

This is a preliminary documentation.

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

Most keys are self-explanatory.

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

**version** of the ini file

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

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

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

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

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

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

## 7 Tools

1 <<version=3.43.1993>>

2 <<date=2020/04/30>>

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

We define some basic macros which just make the code cleaner. `\bbl@add` is now used internally instead of `\addto` because of the unpredictable behavior of the latter. Used in `babel.def` and in `babel.sty`, which means in  $\LaTeX$  is executed twice, but we need them when defining options and `babel.def` cannot be 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@c1#1{\csname bbl@#1\language\endcsname}
14 \def\bbl@loop#1#2#3{\bbl@loop#1{#3}#2,\@nnil,}
15 \def\bbl@loopx#1#2{\expandafter\bbl@loop\expandafter#1\expandafter{#2}}
16 \def\bbl@loop#1#2#3,{%
17   \ifx\@nnil#3\relax\else
18     \def#1{#3}#2\bbl@afterfi\bbl@loop#1{#2}%
19   \fi}
20 \def\bbl@for#1#2#3{\bbl@loopx#1{#2}{\ifx#1\@empty\else#3\fi}}
```

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

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

`\bbl@afterelse` `\bbl@afterfi` Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the `\else` and `\fi` parts of an `\if`-statement<sup>30</sup>. These macros will break if another `\if... \fi` statement appears in one of the arguments and it is not enclosed in braces.

```
27 \long\def\bbl@afterelse#1\else#2\fi{\fi#1}
28 \long\def\bbl@afterfi#1\fi{\fi#1}
```

`\bbl@exp` Now, just syntactical sugar, but it makes partial expansion of some code a lot more simple and readable. Here `\` stands for `\noexpand` and `\<...>` for `\noexpand` applied to a built macro name (the latter does not define the macro if undefined to `\relax`, because it is created locally). The result may be followed by extra arguments, if necessary.

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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).

```

71 \def\bbl@forkv#1#2{%
72   \def\bbl@kvcmd##1##2##3{#2}%
73   \bbl@kvnext#1,\@nil,}
74 \def\bbl@kvnext#1,{%
75   \ifx\@nil#1\relax\else

```

```

76 \bbl@ifblank{#1}{\bbl@forkv@eq#1=@empty=@nil{#1}}%
77 \expandafter\bbl@kvnext
78 \fi}
79 \def\bbl@forkv@eq#1=#2=#3\@nil#4{%
80 \bbl@trim@def\bbl@forkv@a{#1}%
81 \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).

```

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

```

\bbl@replace

```

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

```

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

```

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

```

```

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

```

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

```

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

```

153 <<(*Make sure ProvidesFile is defined)>> ≡
154 \ifx\ProvidesFile\@undefined
155   \def\ProvidesFile#1[#2 #3 #4]{%
156     \wlog{File: #1 #4 #3 <#2>}%
157     \let\ProvidesFile\@undefined}
158 \fi
159 <</Make sure ProvidesFile is defined>>

```

## 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 requires loading `switch.def` in the format.

```

160 <<(*Define core switching macros)>> ≡
161 \ifx\language\@undefined
162   \csname newcount\endcsname\language
163 \fi
164 <</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.

```

165 <<*Define core switching macros>> ≡
166 \ifx\newlanguage\undefined
167   \csname newcount\endcsname\last@language
168   \def\addlanguage#1{%
169     \global\advance\last@language\@ne
170     \ifnum\last@language<\@cclvi
171       \else
172         \errmessage{No room for a new \string\language!}%
173       \fi
174       \global\chardef#1\last@language
175       \wlog{\string#1 = \string\language\the\last@language}}
176   \else
177     \countdef\last@language=19
178     \def\addlanguage{\alloc@9\language\chardef\@cclvi}
179   \fi
180 <</Define core switching macros>>

```

Now we make sure all required files are loaded. When the command `\AtBeginDocument` doesn't exist we assume that we are dealing with a plain-based format or L<sup>A</sup>T<sub>E</sub>X 2.09. In that case the file `plain.def` is needed (which also defines `\AtBeginDocument`, and therefore it is not loaded twice). We need the first part when the format is created, and `\orig@dump` is used as a flag. Otherwise, we need to use the second part, so `\orig@dump` is not defined (`plain.def` undefines it).

Check if the current version of `switch.def` has been previously loaded (mainly, `hyphen.cfg`). If not, load it now. We cannot load `babel.def` here because we first need to declare and process the package options.

## 7.2 The Package File (L<sup>A</sup>T<sub>E</sub>X, `babel.sty`)

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

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

The first two options are for debugging.

```

181 <*package>
182 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
183 \ProvidesPackage{babel}[\<<date>> \<<version>>] The Babel package]
184 \@ifpackagewith{babel}{debug}
185   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
186     \let\bbl@debug\@firstofone}
187   {\providecommand\bbl@trace[1]{}%
188     \let\bbl@debug\@gobble}

```



```

189 <<Basic macros>>
190 % Temporarily repeat here the code for errors
191 \def\bbl@error#1#2{%
192   \begingroup
193     \def\{\MessageBreak}%
194     \PackageError{babel}{#1}{#2}%
195   \endgroup}
196 \def\bbl@warning#1{%
197   \begingroup
198     \def\{\MessageBreak}%
199     \PackageWarning{babel}{#1}%
200   \endgroup}
201 \def\bbl@infowarn#1{%
202   \begingroup
203     \def\{\MessageBreak}%
204     \GenericWarning
205       {(babel) \@spaces\@spaces\@spaces}%
206       {Package babel Info: #1}%
207   \endgroup}
208 \def\bbl@info#1{%
209   \begingroup
210     \def\{\MessageBreak}%
211     \PackageInfo{babel}{#1}%
212   \endgroup}
213 \def\bbl@nocaption{\protect\bbl@nocaption@i}
214 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
215   \global\@namedef{#2}{\textbf{?#1?}}%
216   \@nameuse{#2}%
217   \bbl@warning{%
218     \@backslashchar#2 not set. Please, define\%
219     it in the preamble with something like:\%
220     \string\renewcommand\@backslashchar#2{..}\%
221     Reported}}
222 \def\bbl@tentative{\protect\bbl@tentative@i}
223 \def\bbl@tentative@i#1{%
224   \bbl@warning{%
225     Some functions for '#1' are tentative.\%
226     They might not work as expected and their behavior\%
227     could change in the future.\%
228     Reported}}
229 \def\@nolanerr#1{%
230   \bbl@error
231     {You haven't defined the language #1\space yet.\%
232     Perhaps you misspelled it or your installation\%
233     is not complete}%
234     {Your command will be ignored, type <return> to proceed}}
235 \def\@nopatterns#1{%
236   \bbl@warning
237     {No hyphenation patterns were preloaded for\%
238     the language `#1' into the format.\%
239     Please, configure your TeX system to add them and\%
240     rebuild the format. Now I will use the patterns\%
241     preloaded for \bbl@nulllanguage\space instead}}
242   % End of errors
243 \ifpackagewith{babel}{silent}
244   {\let\bbl@info\@gobble
245    \let\bbl@infowarn\@gobble
246    \let\bbl@warning\@gobble}
247 {}

```

```

248 %
249 \def\AfterBabelLanguage#1{%
250   \global\expandafter\bbbl@add\csname#1.ldf-h@@k\endcsname}%

```

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

```

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

```

### 7.3 base

The first ‘real’ option to be processed is `base`, which set the hyphenation patterns then resets `ver@babel.sty` so that `TeX` forgets about the first loading. After a subset of `babel.def` has been loaded (the old `switch.def`) and `\AfterBabelLanguage` defined, it exits.

Now the `base` option. With it we can define (and load, with `luatex`) hyphenation patterns, even if we are not interested in the rest of `babel`.

```

269 \bbbl@trace{Defining option 'base'}
270 \@ifpackagewith{babel}{base}{%
271   \let\bbbl@onlyswitch\@empty
272   \let\bbbl@provide@locale\relax
273   \input babel.def
274   \let\bbbl@onlyswitch\@undefined
275   \ifx\directlua\@undefined
276     \DeclareOption*{\bbbl@patterns{\CurrentOption}}%
277   \else
278     \input luababel.def
279     \DeclareOption*{\bbbl@patterns@lua{\CurrentOption}}%
280   \fi
281   \DeclareOption{base}{}%
282   \DeclareOption{showlanguages}{}%
283   \ProcessOptions
284   \global\expandafter\let\csname opt@babel.sty\endcsname\relax
285   \global\expandafter\let\csname ver@babel.sty\endcsname\relax
286   \global\let\@ifl@ter@\@ifl@ter
287   \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
288   \endinput}{%
289 % \end{macrocode}
290 %
291 % TODO. Code for lua bidi options must be moved to a logical place. The
292 % problem is |\RequirePackage|, which is forbidden allowed in the options
293 % section.

```

```

294%
295% \begin{macrocode}
296 \ifodd\bb1@engine
297   \def\bb1@activate@preotf{%
298     \let\bb1@activate@preotf\relax % only once
299     \directlua{
300       Babel = Babel or {}
301       %
302       function Babel.pre_otfload_v(head)
303         if Babel.numbers and Babel.digits_mapped then
304           head = Babel.numbers(head)
305         end
306         if Babel.bidi_enabled then
307           head = Babel.bidi(head, false, dir)
308         end
309         return head
310       end
311       %
312       function Babel.pre_otfload_h(head, gc, sz, pt, dir)
313         if Babel.numbers and Babel.digits_mapped then
314           head = Babel.numbers(head)
315         end
316         if Babel.bidi_enabled then
317           head = Babel.bidi(head, false, dir)
318         end
319         return head
320       end
321       %
322       luatexbase.add_to_callback('pre_linebreak_filter',
323         Babel.pre_otfload_v,
324         'Babel.pre_otfload_v',
325       luatexbase.priority_in_callback('pre_linebreak_filter',
326         'luaotfload.node_processor') or nil)
327       %
328       luatexbase.add_to_callback('hpack_filter',
329         Babel.pre_otfload_h,
330         'Babel.pre_otfload_h',
331       luatexbase.priority_in_callback('hpack_filter',
332         'luaotfload.node_processor') or nil)
333     }}
334   \let\bb1@tempa\relax
335   \@ifpackagewith{babel}{bidi=basic}%
336     {\def\bb1@tempa{basic}}%
337     {\@ifpackagewith{babel}{bidi=basic-r}%
338       {\def\bb1@tempa{basic-r}}%
339     }
340   \ifx\bb1@tempa\relax\else
341     \let\bb1@beforeforeign\leavevmode
342     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}%
343     \RequirePackage{luatexbase}%
344     \directlua{
345       require('babel-data-bidi.lua')
346       require('babel-bidi-\bb1@tempa.lua')
347     }
348     \bb1@activate@preotf
349   \fi
350 \fi

```

## 7.4 key=value options and other general option

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

```
351 \bbl@trace{key=value and another general options}
352 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
353 \def\bbl@tempb#1.#2{%
354   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
355 \def\bbl@tempd#1.#2\@nnil{%
356   \ifx\@empty#2%
357     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
358   \else
359     \in@{=}{#1}\ifin@
360     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
361   \else
362     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
363     \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
364   \fi
365 \fi}
366 \let\bbl@tempc\@empty
367 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
368 \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.

```
369 \DeclareOption{KeepShorthandsActive}{}
370 \DeclareOption{activeacute}{}
371 \DeclareOption{activegrave}{}
372 \DeclareOption{debug}{}
373 \DeclareOption{noconfigs}{}
374 \DeclareOption{showlanguages}{}
375 \DeclareOption{silent}{}
376 \DeclareOption{mono}{}
377 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
378 % Don't use. Experimental. TODO.
379 \newif\ifbbl@single
380 \DeclareOption{selectors=off}{\bbl@singletrue}
381 <<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.

```
382 \let\bbl@opt@shorthands\@nnil
383 \let\bbl@opt@config\@nnil
384 \let\bbl@opt@main\@nnil
385 \let\bbl@opt@headfoot\@nnil
386 \let\bbl@opt@layout\@nnil
```

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

```
387 \def\bbl@tempa#1=#2\bbl@tempa{%
388   \bbl@csarg\ifx{opt@#1}\@nnil
```

```

389 \bbl@csarg\edef{opt#1}{#2}%
390 \else
391 \bbl@error
392 {Bad option `#1=#2'. Either you have misspelled the\\%
393 key or there is a previous setting of `#1'. Valid\\%
394 keys are, among others, `shorthands', `main', `bidi',\\%
395 `strings', `config', `headfoot', `safe', `math'.}%
396 {See the manual for further details.}
397 \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.

```

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

```

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

```

406 \ProcessOptions*

```

## 7.5 Conditional loading of shorthands

If there is no shorthands=<chars>, the original babel macros are left untouched, but if there is, these macros are wrapped (in babel.def) to define only those given. A bit of optimization: if there is no shorthands=, then \bbl@ifshorthand is always true, and it is always false if shorthands is empty. Also, some code makes sense only with shorthands=...

```

407 \bbl@trace{Conditional loading of shorthands}
408 \def\bbl@sh@string#1{%
409 \ifx#1\@empty\else
410 \ifx#1t\string~%
411 \else\ifx#1c\string,%
412 \else\string#1%
413 \fi\fi
414 \expandafter\bbl@sh@string
415 \fi}
416 \ifx\bbl@opt@shorthands\@nnil
417 \def\bbl@ifshorthand#1#2#3{#2}%
418 \else\ifx\bbl@opt@shorthands\@empty
419 \def\bbl@ifshorthand#1#2#3{#3}%
420 \else

```

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

```

421 \def\bbl@ifshorthand#1{%
422 \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
423 \ifin@
424 \expandafter\@firstoftwo
425 \else
426 \expandafter\@secondoftwo
427 \fi}

```

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

```
428 \edef\bbl@opt@shorthands{%
429   \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.

```
430 \bbl@ifshorthand{'}%
431   {\PassOptionsToPackage{activeacute}{babel}}{}
432 \bbl@ifshorthand{'}%
433   {\PassOptionsToPackage{activegrave}{babel}}{}
434 \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.

```
435 \ifx\bbl@opt@headfoot\@nnil\else
436   \g@addto@macro\@resetactivechars{%
437     \set@typeset@protect
438     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
439     \let\protect\noexpand}
440 \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.

```
441 \ifx\bbl@opt@safe\@undefined
442   \def\bbl@opt@safe{BR}
443 \fi
444 \ifx\bbl@opt@main\@nnil\else
445   \edef\bbl@language@opts{%
446     \ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
447     \bbl@opt@main}
448 \fi
```

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

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

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

```
461 \input babel.def
```

## 7.6 Cross referencing macros

The  $\LaTeX$  book states:

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

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

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

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

```
462 <<*More package options>> ≡
463 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
464 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
465 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
466 <</More package options>>
```

`\@newl@bel` First we open a new group to keep the changed setting of `\protect` local and then we set the `@safe@actives` switch to true to make sure that any shorthand that appears in any of the arguments immediately expands to its non-active self.

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

```
477 \CheckCommand*\@testdef[3]{%
478   \def\reserved@a{#3}%
479   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
480   \else
481     \@tempswatrue
482   \fi}
```

Now that we made sure that `\@testdef` still has the same definition we can rewrite it. First we make the shorthands ‘safe’. Then we use `\bbl@tempa` as an ‘alias’ for the macro that contains the label which is being checked. Then we define `\bbl@tempb` just as `\@newl@bel` does it. When the label is defined we replace the definition of `\bbl@tempa` by its meaning. If the label didn’t change, `\bbl@tempa` and `\bbl@tempb` should be identical macros.

```
483 \def\@testdef#1#2#3{% TODO. With @samestring?
484   \@safe@activestrue
485   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
486   \def\bbl@tempb{#3}%
487   \@safe@activesfalse
488   \ifx\bbl@tempa\relax
489   \else
490     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
491   \fi
492   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
493   \ifx\bbl@tempa\bbl@tempb
494   \else
495     \@tempswatrue
496   \fi}
497 \fi
```

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

```

498 \bbl@xin@{R}\bbl@opt@safe
499 \ifin@
500   \bbl@redefineroast\ref#1{%
501     \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
502   \bbl@redefineroast\pageref#1{%
503     \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
504 \else
505   \let\org@ref\ref
506   \let\org@pageref\pageref
507 \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.

```

508 \bbl@xin@{B}\bbl@opt@safe
509 \ifin@
510   \bbl@redefine\@citex[#1]#2{%
511     \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
512     \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.

```

513 \AtBeginDocument{%
514   \@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.)

```

515   \def\@citex[#1][#2]#3{%
516     \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
517     \org@citex[#1][#2]{\@tempa}}%
518   }{}

```

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

```

519 \AtBeginDocument{%
520   \@ifpackageloaded{cite}{%
521     \def\@citex[#1]#2{%
522       \@safe@activetrue\org@citex[#1][#2]\@safe@activesfalse}%
523     }{}

```

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

```

524 \bbl@redefine\nocite#1{%
525   \@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 `\biblecite` is needed we define `\biblecite` in such a way that it redefines itself with the proper definition. We call `\bbl@cite@choice` to select the proper definition for `\biblecite`. This new definition is then activated.

```
526 \bbl@redefine\biblecite{%
527   \bbl@cite@choice
528   \biblecite}
```

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

```
529 \def\bbl@biblecite#1#2{%
530   \org@biblecite{#1}{\@safe@activesfalse#2}}
```

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

```
531 \def\bbl@cite@choice{%
532   \global\let\biblecite\bbl@biblecite
533   \@ifpackageloaded{natbib}{\global\let\biblecite\org@biblecite}{}%
534   \@ifpackageloaded{cite}{\global\let\biblecite\org@biblecite}{}%
535   \global\let\bbl@cite@choice\relax}
```

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

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

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

```
537 \bbl@redefine\@bibitem#1{%
538   \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
539 \else
540   \let\org@nocite\nocite
541   \let\org@@citex\citex
542   \let\org@biblecite\biblecite
543   \let\org@@bibitem\@bibitem
544 \fi
```

## 7.7 Marks

`\markright` Because the output routine is asynchronous, we must pass the current language attribute to the head lines. To achieve this we need to adapt the definition of `\markright` and `\markboth` somewhat. However, headlines and footlines can contain text outside marks; for that we must take some actions in the output routine if the ‘headfoot’ options is used. We need to make some redefinitions to the output routine to avoid an endless loop and to correctly handle the page number in bidi documents.

```
545 \bbl@trace{Marks}
546 \IfBabelLayout{sectioning}
547   {\ifx\bbl@opt@headfoot\@nnil
548     \g@addto@macro\@resetactivechars{%
549       \set@typeset@protect
550       \expandafter\select@language@x\expandafter{\bbl@main@language}%
551       \let\protect\noexpand
552       \edef\thepage{% TODO. Only with bidi. See also above
553         \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
554     \fi}
```

```

555 {\ifbbl@single\else
556   \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
557   \markright#1{%
558     \bbl@ifblank{#1}%
559     {\org@markright{}}}%
560     {\toks@{#1}%
561     \bbl@exp{%
562       \org@markright{\protect\foreignlanguage{\language}%
563         {\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. (As of Oct 2019, L<sup>A</sup>T<sub>E</sub>X stores the definition in an intermediate macro, so it's not necessary anymore, but it's preserved for older versions.)

```

564   \ifx\@mkboth\markboth
565     \def\bbl@tempc{\let\@mkboth\markboth}
566   \else
567     \def\bbl@tempc{}
568   \fi
569   \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
570   \markboth#1#2{%
571     \protected@edef\bbl@tempb##1{%
572       \protect\foreignlanguage
573       {\language}{\protect\bbl@restore@actives##1}}%
574     \bbl@ifblank{#1}%
575     {\toks@{}}%
576     {\toks@\expandafter{\bbl@tempb{#1}}}%
577     \bbl@ifblank{#2}%
578     {\@temptokena{}}%
579     {\@temptokena\expandafter{\bbl@tempb{#2}}}%
580     \bbl@exp{\org@markboth{\the\toks@}{\the\@temptokena}}
581     \bbl@tempc
582   \fi} % end ifbbl@single, end \IfBabelLayout

```

## 7.8 Preventing clashes with other packages

### 7.8.1 ifthen

\ifthenelse Sometimes a document writer wants to create a special effect depending on the page a certain fragment of text appears on. This can be achieved by the following piece of code:

```

\ifthenelse{\isodd{\pageref{some:label}}}
{code for odd pages}
{code for even pages}

```

In order for this to work the argument of \isodd needs to be fully expandable. With the above redefinition of \pageref it is not in the case of this example. To overcome that, we add some code to the definition of \ifthenelse to make things work.

We want to revert the definition of \pageref and \ref to their original definition for the first argument of \ifthenelse, so we first need to store their current meanings.

Then we can set the \@safe@actives switch and call the original \ifthenelse. In order to be able to use shorthands in the second and third arguments of \ifthenelse the resetting of the switch *and* the definition of \pageref happens inside those arguments.

```

583 \bbl@trace{Preventing clashes with other packages}

```

```

584 \bbl@xin@{R}\bbl@opt@safe
585 \ifin@
586 \AtBeginDocument{%
587   \ifpackageloaded{ifthen}{%
588     \bbl@redefine@long\ifthenelse#1#2#3{%
589       \let\bbl@temp@pref\pageref
590       \let\pageref\org@pageref
591       \let\bbl@temp@ref\ref
592       \let\ref\org@ref
593       \@safe@activetrue
594       \org@ifthenelse{#1}%
595       {\let\pageref\bbl@temp@pref
596        \let\ref\bbl@temp@ref
597        \@safe@activetrue
598        #2}%
599       {\let\pageref\bbl@temp@pref
600        \let\ref\bbl@temp@ref
601        \@safe@activetrue
602        #3}%
603     }%
604   }{}%
605 }

```

### 7.8.2 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`.  
`\vrefpagemum` The same needs to happen for `\vrefpagemum`.  
`\Ref`

```

606 \AtBeginDocument{%
607   \ifpackageloaded{varioref}{%
608     \bbl@redefine\@@vpageref#1[#2]#3{%
609       \@safe@activetrue
610       \org@@@vpageref{#1}[#2]{#3}%
611       \@safe@activetrue}%
612     \bbl@redefine\vrefpagemum#1#2{%
613       \@safe@activetrue
614       \org@vrefpagemum{#1}{#2}%
615       \@safe@activetrue}%

```

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.

```

616   \expandafter\def\csname Ref \endcsname#1{%
617     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
618   }{}%
619 }
620 \fi

```

### 7.8.3 hline

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

```

621 \AtEndOfPackage{%
622   \AtBeginDocument{%
623     \ifpackageloaded{hhline}%
624       {\expandafter\ifx\csname normal@char\string\endcsname\relax
625         \else
626           \makeatletter
627           \def\@currname{hhline}\input{hhline.sty}\makeatother
628           \fi}%
629     {}}}
```

#### 7.8.4 hyperref

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

```

630 \AtBeginDocument{%
631   \ifx\pdfstringdefDisableCommands\undefined\else
632     \pdfstringdefDisableCommands{\languageshorthands{system}}%
633   \fi}
```

#### 7.8.5 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.

```

634 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
635   \lowercase{\foreignlanguage{#1}}}
```

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

```

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

### 7.9 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of `TEX` and `ℒTEX` always come out in the right encoding. There is a list of non-ASCII encodings.

Unfortunately, fontenc deletes its package options, so we must guess which encodings has been loaded by traversing \@filelist to search for <enc>enc.def. If a non-ASCII has been loaded, we define versions of \TeX and \LaTeX for them using \ensureascii. The default ASCII encoding is set, too (in reverse order): the “main” encoding (when the document begins), the last loaded, or OT1.

\ensureascii

```

655 \bbl@trace{Encoding and fonts}
656 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,PU,PD1}
657 \newcommand\BabelNonText{TS1,T3,TS3}
658 \let\org@TeX\TeX
659 \let\org@LaTeX\LaTeX
660 \let\ensureascii\@firstofone
661 \AtBeginDocument{%
662   \in@false
663   \bbl@foreach\BabelNonASCII{% is there a text non-ascii enc?
664     \ifin@
665       \lowercase{\bbl@xin@{, #1 enc.def, }{, \@filelist, }}%
666       \fi}%
667   \ifin@ % if a text non-ascii has been loaded
668     \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
669     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
670     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
671     \def\bbl@tempb#1\@@{\uppercase{\bbl@tempc#1}ENC.DEF\empty\@@}%
672     \def\bbl@tempc#1ENC.DEF#2\@@{%
673       \ifx\empty#2\else
674         \bbl@ifunset{T@#1}%
675         {}%
676         {\bbl@xin@{, #1, }{, \BabelNonASCII, \BabelNonText, }}%
677         \ifin@
678           \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
679           \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
680         \else
681           \def\ensureascii##1{{\fontencoding{#1}\selectfont##1}}%
682           \fi}%
683     \fi}%
684   \bbl@foreach\@filelist{\bbl@tempb#1\@@}% TODO - \@@ de mas??
685   \bbl@xin@{, \cf@encoding, }{, \BabelNonASCII, \BabelNonText,}%
686   \ifin@
687     \edef\ensureascii#1{%
688       \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
689   \fi
690 \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.

```

691 \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.

```

692 \AtBeginDocument{%

```

```

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

```

716 \DeclareRobustCommand{\latintext}{%
717   \fontencoding{\latinencoding}\selectfont
718   \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.

```

719 \ifx\@undefined\DeclareTextFontCommand
720   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
721 \else
722   \DeclareTextFontCommand{\textlatin}{\latintext}
723 \fi

```

## 7.10 Basic bidi support

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

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

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

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

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

```

724 \bbl@trace{Basic (internal) bidi support}
725 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
726 \def\bbl@rscripts{%
727   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
728   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
729   Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
730   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
731   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
732   Old South Arabian,}%
733 \def\bbl@provide@dirs#1{%
734   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
735   \ifin@
736     \global\bbl@csarg\chardef{wdir@#1}\@ne
737     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
738     \ifin@
739       \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
740     \fi
741   \else
742     \global\bbl@csarg\chardef{wdir@#1}\z@
743   \fi
744   \ifodd\bbl@engine
745     \bbl@csarg\ifcase{wdir@#1}%
746       \directlua{ Babel.locale_props[\the\localeid].texmdir = 'l' }%
747     \or
748       \directlua{ Babel.locale_props[\the\localeid].texmdir = 'r' }%
749     \or
750       \directlua{ Babel.locale_props[\the\localeid].texmdir = 'al' }%
751     \fi
752   \fi}
753 \def\bbl@switchdir{%
754   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
755   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
756   \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}}%
757 \def\bbl@setdirs#1{% TODO - math
758   \ifcase\bbl@select@type % TODO - strictly, not the right test
759     \bbl@bodydir{#1}%
760     \bbl@pardir{#1}%
761   \fi
762   \bbl@texmdir{#1}}
763 \ifodd\bbl@engine % luatex=1
764   \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
765   \DisableBabelHook{babel-bidi}
766   \chardef\bbl@thetexmdir\z@
767   \chardef\bbl@thepardir\z@
768   \def\bbl@getluadir#1{%
769     \directlua{
770       if tex.#1dir == 'TLT' then
771         tex.sprint('0')
772       elseif tex.#1dir == 'TRT' then
773         tex.sprint('1')
774       end}}
775   \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\texmdir.. 3=0 lr/1 rl
776     \ifcase#3\relax
777     \ifcase\bbl@getluadir{#1}\relax\else

```

```

778     #2 TLT\relax
779     \fi
780   \else
781     \ifcase\bbbl@getluadir{#1}\relax
782       #2 TRT\relax
783     \fi
784   \fi}
785 \def\bbbl@textdir#1{%
786   \bbbl@setluadir{text}\textdir{#1}%
787   \chardef\bbbl@thetextdir#1\relax
788   \setattribute\bbbl@attr@dir{\numexpr\bbbl@thepardir*3+#1}}
789 \def\bbbl@pardir#1{%
790   \bbbl@setluadir{par}\pardir{#1}%
791   \chardef\bbbl@thepardir#1\relax}
792 \def\bbbl@bodydir{\bbbl@setluadir{body}\bodydir}
793 \def\bbbl@pagedir{\bbbl@setluadir{page}\pagedir}
794 \def\bbbl@dirparastext{\pardir\the\textdir\relax}%   %%%
795 % Sadly, we have to deal with boxes in math with basic.
796 % Activated every math with the package option bidi=:
797 \def\bbbl@mathboxdir{%
798   \ifcase\bbbl@thetextdir\relax
799     \everyhbox{\textdir TLT\relax}%
800   \else
801     \everyhbox{\textdir TRT\relax}%
802   \fi}
803 \else % pdftex=0, xetex=2
804   \AddBabelHook{babel-bidi}{afterextras}{\bbbl@switchdir}
805   \DisableBabelHook{babel-bidi}
806   \newcount\bbbl@dirlevel
807   \chardef\bbbl@thetextdir\z@
808   \chardef\bbbl@thepardir\z@
809   \def\bbbl@textdir#1{%
810     \ifcase#1\relax
811       \chardef\bbbl@thetextdir\z@
812       \bbbl@textdir@i\beginL\endL
813     \else
814       \chardef\bbbl@thetextdir\@ne
815       \bbbl@textdir@i\beginR\endR
816     \fi}
817   \def\bbbl@textdir@i#1#2{%
818     \ifhmode
819       \ifnum\currentgrouplevel>\z@
820         \ifnum\currentgrouplevel=\bbbl@dirlevel
821           \bbbl@error{Multiple bidi settings inside a group}%
822           {I'll insert a new group, but expect wrong results.}%
823           \bgroup\aftergroup#2\aftergroup\egroup
824         \else
825           \ifcase\currentgroup\or % 0 bottom
826             \aftergroup#2% 1 simple {}
827           \or
828             \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
829           \or
830             \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
831           \or\or\or % vbox vtop align
832           \or
833             \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
834           \or\or\or\or\or\or % output math disc insert vcent mathchoice
835           \or
836             \aftergroup#2% 14 \begingroup

```



```

837         \else
838         \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
839         \fi
840         \fi
841         \bbl@dirlevel\currentgrouplevel
842     \fi
843     #1%
844 \fi}
845 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
846 \let\bbl@bodydir\@gobble
847 \let\bbl@pagedir\@gobble
848 \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).

```

849 \def\bbl@xebidipar{%
850     \let\bbl@xebidipar\relax
851     \TeXeTstate\@ne
852     \def\bbl@xeeverypar{%
853         \ifcase\bbl@thepardir
854         \ifcase\bbl@thetextdir\else\beginR\fi
855         \else
856         {\setbox\z@\lastbox\beginR\box\z@}%
857         \fi}%
858     \let\bbl@severypar\everypar
859     \newtoks\everypar
860     \everypar=\bbl@severypar
861     \bbl@severypar{\bbl@xeeverypar\the\everypar}}
862 \def\bbl@tempb{%
863     \let\bbl@textdir\i\@gobbletwo
864     \let\bbl@xebidipar\@empty
865     \AddBabelHook{bidi}{foreign}{%
866         \def\bbl@tempa{\def\BabelText#####1}%
867         \ifcase\bbl@thetextdir
868         \expandafter\bbl@tempa\expandafter{\BabelText{\LR{#####1}}}%
869         \else
870         \expandafter\bbl@tempa\expandafter{\BabelText{\RL{#####1}}}%
871         \fi}
872     \def\bbl@pardir##1{\ifcase##1\relax\setLR\else\setRL\fi}}
873 \@ifpackagewith{babel}{bidi=bidi}{\bbl@tempb}{}%
874 \@ifpackagewith{babel}{bidi=bidi-l}{\bbl@tempb}{}%
875 \@ifpackagewith{babel}{bidi=bidi-r}{\bbl@tempb}{}%
876 \fi

```

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

```

877 \DeclareRobustCommand\babelsublr[1]{\leavevmode\bbl@textdir\z@#1}}
878 \AtBeginDocument{%
879     \ifx\pdfstringdefDisableCommands\undefined\else
880     \ifx\pdfstringdefDisableCommands\relax\else
881     \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
882     \fi
883 \fi}

```

## 7.11 Local Language Configuration

`\loadlocalcfg` At some sites it may be necessary to add site-specific actions to a language definition file. This can be done by creating a file with the same name as the language definition file, but

with the extension .cfg. For instance the file norsk.cfg will be loaded when the language definition file norsk.ldf is loaded.

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

```

884 \bbl@trace{Local Language Configuration}
885 \ifx\loadlocalcfg\undefined
886   \@ifpackagewith{babel}{noconfigs}%
887   {\let\loadlocalcfg@gobble}%
888   {\def\loadlocalcfg#1{%
889     \InputIfFileExists{#1.cfg}%
890     {\typeout{*****^J%
891               * Local config file #1.cfg used^^J%
892               *}}%
893     \@empty}}
894 \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. TODO. Necessary? Correct place? Used by some ldf file?

```

895 \ifx\@unexpandable@protect\undefined
896   \def\@unexpandable@protect{\noexpand\protect\noexpand}
897   \long\def\protected@write#1#2#3{%
898     \begingroup
899       \let\thepage\relax
900       #2%
901       \let\protect\@unexpandable@protect
902       \edef\reserved@a{\write#1{#3}}%
903       \reserved@a
904     \endgroup
905     \if@nobreak\ifvmode\nobreak\fi\fi}
906 \fi
907 %
908 % \subsection{Language options}
909 %
910 %   Languages are loaded when processing the corresponding option
911 %   \textit{except} if a |main| language has been set. In such a
912 %   case, it is not loaded until all options has been processed.
913 %   The following macro inputs the ldf file and does some additional
914 %   checks (|\input| works, too, but possible errors are not caught).
915 %
916 %   \begin{macrocode}
917 \bbl@trace{Language options}
918 \let\bbl@afterlang\relax
919 \let\BabelModifiers\relax
920 \let\bbl@loaded\@empty
921 \def\bbl@load@language#1{%
922   \InputIfFileExists{#1.ldf}%
923   {\edef\bbl@loaded{\CurrentOption
924     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
925     \expandafter\let\expandafter\bbl@afterlang
926       \csname\CurrentOption.ldf-h@@k\endcsname
927     \expandafter\let\expandafter\BabelModifiers
928       \csname bbl@mod@\CurrentOption\endcsname}%
929   {\bbl@error{%
930     Unknown option '\CurrentOption'. Either you misspelled it\\%
931     or the language definition file \CurrentOption.ldf was not found}}%
932     Valid options are: shorthands=, KeepShorthandsActive,\\%
933     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
934     headfoot=, strings=, config=, hyphenmap=, or a language name.}}}

```

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

```

935 \def\bbl@try@load@lang#1#2#3{%
936   \IfFileExists{\CurrentOption.ldf}%
937     {\bbl@load@language{\CurrentOption}}%
938     {\bbl@load@language{#2}#3}}
939 \DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}}
940 \DeclareOption{brazil}{\bbl@try@load@lang{}{portuges}}
941 \DeclareOption{brazilian}{\bbl@try@load@lang{}{portuges}}
942 \DeclareOption{hebrew}{%
943   \input{rlbabel.def}%
944   \bbl@load@language{hebrew}}
945 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}}
946 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}}
947 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}}
948 \DeclareOption{polutonikogreek}{%
949   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
950 \DeclareOption{portuguese}{\bbl@try@load@lang{}{portuges}}
951 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}}
952 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}}
953 \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.

```

954 \ifx\bbl@opt@config\@nnil
955   \ifpackagewith{babel}{noconfigs}{%
956     {\InputIfFileExists{bblopts.cfg}%
957       {\typeout{*****^J%
958         * Local config file bblopts.cfg used^^J%
959         *}}}%
960     }%
961   \else
962     \InputIfFileExists{\bbl@opt@config.cfg}%
963     {\typeout{*****^J%
964       * Local config file \bbl@opt@config.cfg used^^J%
965       *}}%
966     {\bbl@error{%
967       Local config file '\bbl@opt@config.cfg' not found}%
968       Perhaps you misspelled it.}}%
969   \fi

```

Recognizing global options in packages not having a closed set of them is not trivial, as for them to be processed they must be defined explicitly. So, package options not yet taken into account and stored in `bbl@language@opts` are assumed to be languages (note this list also contains the language given with `main`). If not declared above, the names of the option and the file are the same.

```

970 \bbl@for\bbl@tempa\bbl@language@opts{%
971   \bbl@ifunset{ds@\bbl@tempa}%
972     {\edef\bbl@tempb{%
973       \noexpand\DeclareOption
974       {\bbl@tempa}%
975       {\noexpand\bbl@load@language{\bbl@tempa}}}%
976     \bbl@tempb}%
977   \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.

```

978 \bbl@foreach\@classoptionslist{%
979   \bbl@ifunset{ds@#1}%
980     {\IfFileExists{#1.ldf}%
981       {\DeclareOption{#1}{\bbl@load@language{#1}}}%
982       {}}%
983   {}}

```

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

```

984 \ifx\bbl@opt@main\@nnil\else
985   \expandafter
986   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
987   \DeclareOption{\bbl@opt@main}{}
988 \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):

```

989 \def\AfterBabelLanguage#1{%
990   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
991 \DeclareOption*{}
992 \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.

```

993 \bbl@trace{Option 'main'}
994 \ifx\bbl@opt@main\@nnil
995   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
996   \let\bbl@tempc\@empty
997   \bbl@for\bbl@tempb\bbl@tempa{%
998     \bbl@xin@{\bbl@tempb,}{,\bbl@loaded,}%
999     \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
1000   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
1001   \expandafter\bbl@tempa\bbl@loaded,\@nnil
1002   \ifx\bbl@tempb\bbl@tempc\else
1003     \bbl@warning{%
1004       Last declared language option is '\bbl@tempc',\%
1005       but the last processed one was '\bbl@tempb'.\%
1006       The main language cannot be set as both a global\%
1007       and a package option. Use 'main=\bbl@tempc' as\%
1008       option. Reported}%
1009   \fi
1010 \else
1011   \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
1012   \ExecuteOptions{\bbl@opt@main}
1013   \DeclareOption*{}
1014   \ProcessOptions*
1015 \fi
1016 \def\AfterBabelLanguage{%
1017   \bbl@error
1018     {Too late for \string\AfterBabelLanguage}%
1019     {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.

```

1020 \ifx\bbl@main@language\@undefined
1021   \bbl@info{%
1022     You haven't specified a language. I'll use 'nil'\%
1023     as the main language. Reported}
1024   \bbl@load@language{nil}
1025 \fi
1026 \</package>
1027 \*core>

```

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

The kernel of the babel system is currently stored in `babel.def`. The file `babel.def` contains most of the code. The file `hyphen.cfg` is a file that can be loaded into the format, which is necessary when you want to be able to switch hyphenation patterns. Because plain  $\TeX$  users might want to use some of the features of the babel system too, care has to be taken that plain  $\TeX$  can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain  $\TeX$  and  $\LaTeX$ , some of it is for the  $\LaTeX$  case only. Plain formats based on `etex` (`etex`, `xetex`, `luatex`) don't load `hyphen.cfg` but `etex.src`, which follows a different naming convention, so we need to define the babel names. It presumes `language.def` exists and it is the same file used when formats were created.

### 8.1 Tools

```

1028 \ifx\ldf@quit\@undefined\else
1029 \endinput\fi % Same line!
1030 \<<Make sure ProvidesFile is defined>>
1031 \ProvidesFile{babel.def}[\<<date>> \<<version>> Babel common definitions]

```

The file `babel.def` expects some definitions made in the  $\LaTeX 2_{\epsilon}$  style file. So, In  $\LaTeX 2.09$  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 are 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).

```

1032 \ifx\AtBeginDocument\@undefined % TODO. change test.
1033   \<<Emulate LaTeX>>
1034   \def\language{english}%
1035   \let\bbl@opt@shorthands\@nnil
1036   \def\bbl@ifshorthand#1#2#3{#2}%
1037   \let\bbl@language@opts\@empty
1038   \ifx\babeloptionstrings\@undefined
1039     \let\bbl@opt@strings\@nnil
1040   \else
1041     \let\bbl@opt@strings\babeloptionstrings
1042   \fi
1043   \def\BabelStringsDefault{generic}
1044   \def\bbl@tempa{normal}
1045   \ifx\babeloptionmath\bbl@tempa
1046     \def\bbl@mathnormal{\noexpand\textormath}
1047   \fi
1048   \def\AfterBabelLanguage#1#2{}
1049   \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi

```

```

1050 \let\bbl@afterlang\relax
1051 \def\bbl@opt@safe{BR}
1052 \ifx\uclclist\@undefined\let\uclclist\@empty\fi
1053 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
1054 \expandafter\newif\csname ifbbl@single\endcsname
1055 \fi

```

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

```

1056 \ifx\bbl@trace\@undefined
1057 \let\LdfInit\endinput
1058 \def\ProvidesLanguage#1{\endinput}
1059 \endinput\fi % Same line!

```

And continue.

## 9 Multiple languages

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

Plain T<sub>E</sub>X version 3.0 provides the primitive `\language` that is used to store the current language. When used with a pre-3.0 version this function has to be implemented by allocating a counter.

```

1060 <<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.

```

1061 \def\bbl@version{<<version>>}
1062 \def\bbl@date{<<date>>}
1063 \def\adddialect#1#2{%
1064   \global\chardef#1#2\relax
1065   \bbl@usehooks{adddialect}{#1}{#2}}%
1066 \begingroup
1067   \count@#1\relax
1068   \def\bbl@elt##1##2##3##4{%
1069     \ifnum\count@=##2\relax
1070       \bbl@info{\string#1 = using hyphenrules for ##1\\%
1071         (\string\language\the\count@)}%
1072       \def\bbl@elt####1####2####3####4}%
1073     \fi}%
1074   \bbl@cs{languages}%
1075 \endgroup

```

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

```

1076 \def\bbl@fixname#1{%
1077   \begingroup
1078   \def\bbl@tempe{l@}%
1079   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
1080   \bbl@tempd
1081   {\lowercase\expandafter{\bbl@tempd}%
1082    {\uppercase\expandafter{\bbl@tempd}%
1083     \@empty
1084     {\edef\bbl@tempd{\def\noexpand#1{#1}}}%

```

```

1085         \uppercase\expandafter{\bbl@tempd}}}%
1086     {\edef\bbl@tempd{\def\noexpand#1{#1}}}%
1087     \lowercase\expandafter{\bbl@tempd}}}%
1088     \@empty
1089     \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
1090     \bbl@tempd
1091     \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}%
1092 \def\bbl@iflanguage#1{%
1093     \ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}

```

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

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

```

1094 \def\bbl@bcpcase#1#2#3#4\@#5{%
1095     \ifx\@empty#3%
1096         \uppercase{\def#5{#1#2}}}%
1097     \else
1098         \uppercase{\def#5{#1}}}%
1099         \lowercase{\edef#5{#5#2#3#4}}}%
1100     \fi}
1101 \def\bbl@bcpllookup#1-#2-#3-#4\@{%
1102     \let\bbl@bcp\relax
1103     \lowercase{\def\bbl@tempa{#1}}}%
1104     \ifx\@empty#2%
1105         \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1106     \else\ifx\@empty#3%
1107         \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
1108         \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
1109         {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}}%
1110         {}%
1111         \ifx\bbl@bcp\relax
1112             \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1113         \fi
1114     \else
1115         \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
1116         \bbl@bcpcase#3\@empty\@empty\@{\bbl@tempc
1117         \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
1118         {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}}%
1119         {}%
1120         \ifx\bbl@bcp\relax
1121             \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1122             {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}}%
1123             {}%
1124         \fi
1125         \ifx\bbl@bcp\relax
1126             \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1127             {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}}%
1128             {}%
1129         \fi
1130         \ifx\bbl@bcp\relax
1131             \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1132         \fi
1133     \fi\fi}
1134 \let\bbl@autoload@options\@empty

```

```

1135 \let\bbl@initoload\relax
1136 \def\bbl@provide@locale{%
1137   \ifx\babelprovide\undefined
1138     \bbl@error{For a language to be defined on the fly 'base'\\%
1139       is not enough, and the whole package must be\\%
1140       loaded. Either delete the 'base' option or\\%
1141       request the languages explicitly}%
1142     {See the manual for further details.}%
1143   \fi
1144 % TODO. Option to search if loaded, with \LocaleForEach
1145 \let\bbl@auxname\language % Still necessary. TODO
1146 \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
1147 {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
1148 \ifbbl@bcpallowed
1149   \expandafter\ifx\csname date\language\endcsname\relax
1150     \expandafter
1151     \bbl@bcplookup\language-\@empty-\@empty-\@empty\@@
1152     \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
1153       \edef\language{\bbl@bcp@prefix\bbl@bcp}%
1154       \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
1155       \expandafter\ifx\csname date\language\endcsname\relax
1156         \let\bbl@initoload\bbl@bcp
1157         \bbl@exp{\bbl@babelprovide[\bbl@autoload@bcptoptions]{\language}}%
1158         \let\bbl@initoload\relax
1159       \fi
1160       \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
1161     \fi
1162   \fi
1163 \fi
1164 \expandafter\ifx\csname date\language\endcsname\relax
1165   \IfFileExists{babel-\language.tex}%
1166   {\bbl@exp{\bbl@babelprovide[\bbl@autoload@options]{\language}}}%
1167   {}%
1168 \fi}

```

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

```

1169 \def\iflanguage#1{%
1170   \bbl@iflanguage{#1}%
1171   \ifnum\csname l@#1\endcsname=\language
1172     \expandafter\@firstoftwo
1173   \else
1174     \expandafter\@secondoftwo
1175   \fi}}

```

## 9.1 Selecting the language

`\selectlanguage` The macro `\selectlanguage` checks whether the language is already defined before it performs its actual task, which is to update `\language` and activate language-specific definitions.

```

1176 \let\bbl@select@type\z@
1177 \edef\selectlanguage{%
1178   \noexpand\protect
1179   \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`.

```
1180 \ifx\@undefined\protect\let\protect\relax\fi
```

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

```
1181 \let\xstring\string
```

Since version 3.5 babel writes entries to the auxiliary files in order to typeset table of contents etc. in the correct language environment.

`\bbl@pop@language` But when the language change happens *inside* a group the end of the group doesn't write anything to the auxiliary files. Therefore we need TeX's `aftergroup` mechanism to help us. The command `\aftergroup` stores the token immediately following it to be executed when the current group is closed. So we define a temporary control sequence `\bbl@pop@language` to be executed at the end of the group. It calls `\bbl@set@language` with the name of the current language as its argument.

`\bbl@language@stack` The previous solution works for one level of nesting groups, but as soon as more levels are used it is no longer adequate. For that case we need to keep track of the nested languages using a stack mechanism. This stack is called `\bbl@language@stack` and initially empty.

```
1182 \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`

```
1183 \def\bbl@push@language{%
1184   \ifx\language\@undefined\else
1185     \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1186   \fi}
```

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

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

```
1187 \def\bbl@pop@lang#1+##3{%
1188   \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 TeX first *expands* the stack, stored in `\bbl@language@stack`. The result of that is that the argument string of `\bbl@pop@lang` contains one or more language names, each followed by a '+'-sign (zero language names won't occur as this macro will only be called after something has been pushed on the stack) followed by the '&'-sign and finally the reference to the stack.

```
1189 \let\bbl@ifrestoring\@secondoftwo
1190 \def\bbl@pop@language{%
1191   \expandafter\bbl@pop@lang\bbl@language@stack&\bbl@language@stack
1192   \let\bbl@ifrestoring\@firstoftwo
1193   \expandafter\bbl@set@language\expandafter{\language}%
1194   \let\bbl@ifrestoring\@secondoftwo}
```

Once the name of the previous language is retrieved from the stack, it is fed to `\bbl@set@language` to do the actual work of switching everything that needs switching.

An alternative way to identify languages (in the babel sense) with a numerical value is introduced in 3.30. This is one of the first steps for a new interface based on the concept of locale, which explains the name of `\localeid`. This means `\l@...` will be reserved for hyphenation patterns (so that two locales can share the same rules).

```

1195 \chardef\localeid\z@
1196 \def\bbl@id@last{0} % No real need for a new counter
1197 \def\bbl@id@assign{%
1198   \bbl@ifunset{bbl@id@@\language}%
1199   {\count@bbl@id@last\relax
1200    \advance\count@ne
1201    \bbl@csarg\chardef{id@@\language}\count@
1202    \edef\bbl@id@last{\the\count@}%
1203    \ifcase\bbl@engine\or
1204      \directlua{
1205        Babel = Babel or {}
1206        Babel.locale_props = Babel.locale_props or {}
1207        Babel.locale_props[\bbl@id@last] = {}
1208        Babel.locale_props[\bbl@id@last].name = '\language'
1209      }%
1210    \fi}%
1211  }%
1212  \chardef\localeid\bbl@c{l{id@}}

```

The unprotected part of `\selectlanguage`.

```

1213 \expandafter\def\csname selectlanguage \endcsname#1{%
1214   \ifnum\bbl@hymapsel=\@ccclv\let\bbl@hymapsel\tw@fi
1215   \bbl@push@language
1216   \aftergroup\bbl@pop@language
1217   \bbl@set@language{#1}}

```

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

```

1218 \def\BabelContentsFiles{toc,lof,lot}
1219 \def\bbl@set@language#1{% from selectlanguage, pop@
1220   % The old buggy way. Preserved for compatibility.
1221   \edef\language{%
1222     \ifnum\escapechar=\expandafter`\string#1\@empty
1223     \else\string#1\@empty\fi}%
1224   \ifcat\relax\noexpand#1%
1225     \expandafter\ifx\csname date\language\endcsname\relax
1226       \edef\language{#1}%
1227       \let\localename\language
1228     \else
1229       \bbl@info{Using '\string\language' instead of 'language' is%%
1230         deprecated. If what you want is to use a%%
1231         macro containing the actual locale, make%%
1232         sure it does not not match any language.%%
1233         Reported}%

```

```

1234 %                I'll\\%
1235 %                try to fix '\string\localename', but I cannot promise\\%
1236 %                anything. Reported}%
1237 \ifx\scantokens\@undefined
1238 \def\localename{??}%
1239 \else
1240 \scantokens\expandafter{\expandafter
1241 \def\expandafter\localename\expandafter{\language}%
1242 \fi
1243 \fi
1244 \else
1245 \def\localename{#1}% This one has the correct catcodes
1246 \fi
1247 \select@language{\language}%
1248 % write to auxs
1249 \expandafter\ifx\csname date\language\endcsname\relax\else
1250 \if@filesw
1251 \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
1252 \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
1253 \fi
1254 \bbl@usehooks{write}{}}%
1255 \fi
1256 \fi}
1257 %
1258 \newif\ifbbl@bcpallowed
1259 \bbl@bcpallowedfalse
1260 \def\select@language#1{% from set@, babel@aux
1261 % set hymap
1262 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1263 % set name
1264 \edef\language{#1}%
1265 \bbl@fixname\language
1266 % TODO. name@map must be here?
1267 \bbl@provide@locale
1268 \bbl@iflanguage\language{%
1269 \expandafter\ifx\csname date\language\endcsname\relax
1270 \bbl@error
1271 {Unknown language '\language'. Either you have\\%
1272 misspelled its name, it has not been installed,\\%
1273 or you requested it in a previous run. Fix its name,\\%
1274 install it or just rerun the file, respectively. In\\%
1275 some cases, you may need to remove the aux file}%
1276 {You may proceed, but expect wrong results}%
1277 \else
1278 % set type
1279 \let\bbl@select@type\z@
1280 \expandafter\bbl@switch\expandafter{\language}%
1281 \fi}}
1282 \def\babel@aux#1#2{%
1283 \select@language{#1}%
1284 \bbl@foreach\BabelContentsFiles{%
1285 \@writefile{##1}{\babel@toc{#1}{#2}}}% % TODO - ok in plain?
1286 \def\babel@toc#1#2{%
1287 \select@language{#1}}

```

First, check if the user asks for a known language. If so, update the value of `\language` and call `\originalTeX` to bring  $\TeX$  in a certain pre-defined state.

The name of the language is stored in the control sequence `\language`.

Then we have to *redefine* `\originalTeX` to compensate for the things that have been

activated. To save memory space for the macro definition of `\originalTeX`, we construct the control sequence name for the `\noextras<lang>` command at definition time by expanding the `\csname` primitive.

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

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

```

1288 \newif\ifbbl@usedategroup
1289 \def\bbl@switch#1{% from select@, foreign@
1290 % make sure there is info for the language if so requested
1291 \bbl@ensureinfo{#1}%
1292 % restore
1293 \originalTeX
1294 \expandafter\def\expandafter\originalTeX\expandafter{%
1295 \csname noextras#1\endcsname
1296 \let\originalTeX\@empty
1297 \babel@beginsave}%
1298 \bbl@usehooks{afterreset}{}%
1299 \languageshorthands{none}%
1300 % set the locale id
1301 \bbl@id@assign
1302 % switch captions, date
1303 \ifcase\bbl@select@type
1304 \ifhmode
1305 \hskip\z@skip % trick to ignore spaces
1306 \csname captions#1\endcsname\relax
1307 \csname date#1\endcsname\relax
1308 \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
1309 \else
1310 \csname captions#1\endcsname\relax
1311 \csname date#1\endcsname\relax
1312 \fi
1313 \else
1314 \ifbbl@usedategroup % if \foreign... within \<lang>date
1315 \bbl@usedategroupfalse
1316 \ifhmode
1317 \hskip\z@skip % trick to ignore spaces
1318 \csname date#1\endcsname\relax
1319 \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
1320 \else
1321 \csname date#1\endcsname\relax
1322 \fi
1323 \fi
1324 \fi
1325 % switch extras
1326 \bbl@usehooks{beforeextras}{}%
1327 \csname extras#1\endcsname\relax
1328 \bbl@usehooks{afterextras}{}%
1329 % > babel-ensure
1330 % > babel-sh-<short>
1331 % > babel-bidi
1332 % > babel-fontspec
1333 % hyphenation - case mapping
1334 \ifcase\bbl@opt@hyphenmap\or
1335 \def\BabelLower##1##2{\lccode##1=##2\relax}%

```

```

1336 \ifnum\bbl@hymapsel>4\else
1337 \csname\language @bbl@hyphenmap\endcsname
1338 \fi
1339 \chardef\bbl@opt@hyphenmap\z@
1340 \else
1341 \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
1342 \csname\language @bbl@hyphenmap\endcsname
1343 \fi
1344 \fi
1345 \global\let\bbl@hymapsel@cclv
1346 % hyphenation - patterns
1347 \bbl@patterns{#1}%
1348 % hyphenation - mins
1349 \babel@savevariable\lefthyphenmin
1350 \babel@savevariable\righthyphenmin
1351 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1352 \set@hyphenmins\tw@thr@@\relax
1353 \else
1354 \expandafter\expandafter\expandafter\set@hyphenmins
1355 \csname #1hyphenmins\endcsname\relax
1356 \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.

```

1357 \long\def\otherlanguage#1{%
1358 \ifnum\bbl@hymapsel=\cclv\let\bbl@hymapsel\thr@@\fi
1359 \csname selectlanguage \endcsname{#1}%
1360 \ignorespaces}

```

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

```

1361 \long\def\endotherlanguage{%
1362 \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`.

```

1363 \expandafter\def\csname otherlanguage*\endcsname#1{%
1364 \ifnum\bbl@hymapsel=\cclv\chardef\bbl@hymapsel4\relax\fi
1365 \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”.

```

1366 \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.

```

1367 \providecommand\bbl@beforeforeign{}
1368 \edef\foreignlanguage{%
1369   \noexpand\protect
1370   \expandafter\noexpand\csname foreignlanguage \endcsname}
1371 \expandafter\def\csname foreignlanguage \endcsname{%
1372   \@ifstar\bbl@foreign@s\bbl@foreign@x}
1373 \def\bbl@foreign@x#1#2{%
1374   \begingroup
1375     \let\BabelText\@firstofone
1376     \bbl@beforeforeign
1377     \foreign@language{#1}%
1378     \bbl@usehooks{foreign}{}%
1379     \BabelText{#2}% Now in horizontal mode!
1380   \endgroup}
1381 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@@par
1382   \begingroup
1383     {\par}%
1384     \let\BabelText\@firstofone
1385     \foreign@language{#1}%
1386     \bbl@usehooks{foreign*}{}%
1387     \bbl@dirparastext
1388     \BabelText{#2}% Still in vertical mode!
1389     {\par}%
1390   \endgroup}

```

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

```

1391 \def\foreign@language#1{%
1392   % set name
1393   \edef\language#1%
1394   \bbl@fixname\language
1395   % TODO. name@map here?
1396   \bbl@provide@locale
1397   \bbl@iflanguage\language%
1398     \expandafter\ifx\csname date\language\endcsname\relax
1399     \bbl@warning % TODO - why a warning, not an error?
1400       {Unknown language `#1'. Either you have\\%
1401         misspelled its name, it has not been installed,\\%
1402         or you requested it in a previous run. Fix its name,\\%
1403         install it or just rerun the file, respectively. In\\%
1404         some cases, you may need to remove the aux file.\\%

```

```

1405      I'll proceed, but expect wrong results.\\%
1406      Reported}%
1407      \fi
1408      % set type
1409      \let\bbl@select@type\@ne
1410      \expandafter\bbl@switch\expandafter{\language}%

```

**\bbl@patterns** This macro selects the hyphenation patterns by changing the \language register. If special hyphenation patterns are available specifically for the current font encoding, use them instead of the default.

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

```

1411 \let\bbl@hyphlist\@empty
1412 \let\bbl@hyphenation@relax
1413 \let\bbl@pttnlist\@empty
1414 \let\bbl@patterns@relax
1415 \let\bbl@hymapsel=\@cclv
1416 \def\bbl@patterns#1{%
1417   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1418     \csname l@#1\endcsname
1419     \edef\bbl@tempa{#1}%
1420   \else
1421     \csname l@#1:\f@encoding\endcsname
1422     \edef\bbl@tempa{#1:\f@encoding}%
1423   \fi
1424   \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}%
1425   % > luatex
1426   \@ifundefined{bbl@hyphenation@}{% Can be \relax!
1427     \begingroup
1428       \bbl@xin@{\number\language,}{,\bbl@hyphlist}%
1429     \ifin@else
1430       \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}%
1431       \hyphenation{%
1432         \bbl@hyphenation@
1433         \@ifundefined{bbl@hyphenation@#1}%
1434         \@empty
1435         {\space\csname bbl@hyphenation@#1\endcsname}}%
1436       \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1437     \fi
1438   \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\*.

```

1439 \def\hyphenrules#1{%
1440   \edef\bbl@tempf{#1}%
1441   \bbl@fixname\bbl@tempf
1442   \bbl@iflanguage\bbl@tempf{%
1443     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1444     \languageshorthands{none}%
1445     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1446       \set@hyphenmins\tw@\thr@@\relax

```

```

1447 \else
1448 \expandafter\expandafter\expandafter\set@hyphenmins
1449 \csname\bbl@tempf hyphenmins\endcsname\relax
1450 \fi}}
1451 \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.

```

1452 \def\providehyphenmins#1#2{%
1453 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1454 \namedef{#1hyphenmins}{#2}%
1455 \fi}

```

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

```

1456 \def\set@hyphenmins#1#2{%
1457 \lefthyphenmin#1\relax
1458 \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.

```

1459 \ifx\ProvidesFile\@undefined
1460 \def\ProvidesLanguage#1[#2 #3 #4]{%
1461 \wlog{Language: #1 #4 #3 <#2>}%
1462 }
1463 \else
1464 \def\ProvidesLanguage#1{%
1465 \begingroup
1466 \catcode`\ 10 %
1467 \@makeother\%
1468 \@ifnextchar[%]
1469 {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
1470 \def\@provideslanguage#1[#2]{%
1471 \wlog{Language: #1 #2}%
1472 \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1473 \endgroup}
1474 \fi

```

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

```

1475 \ifx\originalTeX\@undefined\let\originalTeX\@empty\fi

```

Because this part of the code can be included in a format, we make sure that the macro which initializes the save mechanism, `\babel@beginsave`, is not considered to be undefined.

```

1476 \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’:

```

1477 \providecommand\setlocale{%
1478 \bbl@error
1479 {Not yet available}%
1480 {Find an armchair, sit down and wait}}

```



```

1481 \let\uselocale\setlocale
1482 \let\locale\setlocale
1483 \let\selectlocale\setlocale
1484 \let\localename\setlocale
1485 \let\textlocale\setlocale
1486 \let\textlanguage\setlocale
1487 \let\language\setlocale

```

## 9.2 Errors

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

`\@noopterr` When the package was loaded without options not everything will work as expected. An error message is issued in that case.  
 When the format knows about `\PackageError` it must be  $\text{\LaTeX 2}_{\epsilon}$ , so we can safely use its error handling interface. Otherwise we'll have to 'keep it simple'.  
 Infos are not written to the console, but on the other hand many people think warnings are errors, so a further message type is defined: an important info which is sent to the console.

```

1488 \edef\bbl@nulllanguage{\string\language=0}
1489 \ifx\PackageError\@undefined % TODO. Move to Plain
1490 \def\bbl@error#1#2{%
1491   \begingroup
1492     \newlinechar=`^^J
1493     \def\{^^J(babel) }%
1494     \errhelp{#2}\errmessage{\{#1}%
1495   \endgroup}
1496 \def\bbl@warning#1{%
1497   \begingroup
1498     \newlinechar=`^^J
1499     \def\{^^J(babel) }%
1500     \message{\{#1}%
1501   \endgroup}
1502 \let\bbl@infowarn\bbl@warning
1503 \def\bbl@info#1{%
1504   \begingroup
1505     \newlinechar=`^^J
1506     \def\{^^J}%
1507     \wlog{#1}%
1508   \endgroup}
1509 \fi
1510 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1511 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1512   \global\@namedef{#2}{\textbf{?#1?}}%
1513   \@nameuse{#2}%
1514   \bbl@warning{%
1515     \@backslashchar#2 not set. Please, define\%
1516     it in the preamble with something like:\%
1517     \string\renewcommand\@backslashchar#2{..}\%
1518     Reported}}
1519 \def\bbl@tentative{\protect\bbl@tentative@i}
1520 \def\bbl@tentative@i#1{%
1521   \bbl@warning{%
1522     Some functions for '#1' are tentative.\%

```

```

1523   They might not work as expected and their behavior\\%
1524   could change in the future.\\%
1525   Reported}}
1526 \def\nolanerr#1{%
1527   \bbl@error
1528   {You haven't defined the language #1\space yet.\\%
1529   Perhaps you misspelled it or your installation\\%
1530   is not complete}%
1531   {Your command will be ignored, type <return> to proceed}}
1532 \def\nopatterns#1{%
1533   \bbl@warning
1534   {No hyphenation patterns were preloaded for\\%
1535   the language '#1' into the format.\\%
1536   Please, configure your TeX system to add them and\\%
1537   rebuild the format. Now I will use the patterns\\%
1538   preloaded for \bbl@nulllanguage\space instead}}
1539 \let\bbl@usehooks\@gobbletwo
1540 \ifx\bbl@onlyswitch\@empty\endinput\fi
1541 % Here ended switch.def

Here ended switch.def.

1542 \ifx\directlua\@undefined\else
1543   \ifx\bbl@luapatterns\@undefined
1544     \input luababel.def
1545   \fi
1546 \fi
1547 <<Basic macros>>
1548 \bbl@trace{Compatibility with language.def}
1549 \ifx\bbl@languages\@undefined
1550   \ifx\directlua\@undefined
1551     \openin1 = language.def % TODO. Remove hardcoded number
1552     \ifeof1
1553       \closein1
1554       \message{I couldn't find the file language.def}
1555     \else
1556       \closein1
1557       \begingroup
1558         \def\addlanguage#1#2#3#4#5{%
1559           \expandafter\ifx\csname lang@#1\endcsname\relax\else
1560             \global\expandafter\let\csname l@#1\expandafter\endcsname
1561             \csname lang@#1\endcsname
1562           \fi}%
1563         \def\uselanguage#1{%
1564           \input language.def
1565         \endgroup
1566       \fi
1567     \fi
1568     \chardef\l@english\z@
1569 \fi

```

\addto It takes two arguments, a *<control sequence>* and TeX-code to be added to the *<control sequence>*.

If the *<control sequence>* has not been defined before it is defined now. The control sequence could also expand to `\relax`, in which case a circular definition results. The net result is a stack overflow. Note there is an inconsistency, because the assignment in the last branch is global.

```

1570 \def\addto#1#2{%
1571   \ifx#1\@undefined

```

```

1572 \def#1{#2}%
1573 \else
1574 \ifx#1\relax
1575 \def#1{#2}%
1576 \else
1577 {\toks@\expandafter{#1#2}%
1578 \xdef#1{\the\toks@}}%
1579 \fi
1580 \fi}

```

The macro `\initiate@active@char` below takes all the necessary actions to make its argument a shorthand character. The real work is performed once for each character. But first we define a little tool. `TODO`. Always used with additional expansions. Move them here? Move the macro to `basic`?

```

1581 \def\bbl@withactive#1#2{%
1582 \begingroup
1583 \lccode`~=#2\relax
1584 \lowercase{\endgroup#1~}}

```

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

```

1585 \def\bbl@redefine#1{%
1586 \edef\bbl@tempa{\bbl@stripslash#1}%
1587 \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1588 \expandafter\def\csname\bbl@tempa\endcsname{
1589 \@onlypreamble\bbl@redefine

```

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

```

1590 \def\bbl@redefine@long#1{%
1591 \edef\bbl@tempa{\bbl@stripslash#1}%
1592 \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1593 \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname{
1594 \@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_`.

```

1595 \def\bbl@redefineroobust#1{%
1596 \edef\bbl@tempa{\bbl@stripslash#1}%
1597 \bbl@ifunset{\bbl@tempa\space}%
1598 {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1599 \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1600 {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1601 \@namedef{\bbl@tempa\space}%
1602 \@onlypreamble\bbl@redefineroobust

```

### 9.3 Hooks

Admittedly, the current implementation is a somewhat simplistic and does very little to catch errors, but it is meant for developers, after all. `\bbl@usehooks` is the commands used by `babel` to execute hooks defined for an event.

```

1603 \bbl@trace{Hooks}
1604 \newcommand\AddBabelHook[3][{}]{%
1605   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1606   \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1607   \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1608   \bbl@ifunset{bbl@ev@#2@#3@#1}%
1609     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elt{#2}}}%
1610     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1611   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1612 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1613 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1614 \def\bbl@usehooks#1#2{%
1615   \def\bbl@elt##1{%
1616     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@#2}}%
1617     \bbl@cs{ev@#1@}%
1618     \ifx\language\@undefined\else % Test required for Plain (?)
1619       \def\bbl@elt##1{%
1620         \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1@#2}}%
1621         \bbl@cl{ev@#1}%
1622       \fi}

```

To ensure forward compatibility, arguments in hooks are set implicitly. So, if a further argument is added in the future, there is no need to change the existing code. Note events intended for hyphen.cfg are also loaded (just in case you need them for some reason).

```

1623 \def\bbl@evargs{,% <- don't delete this comma
1624   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1625   addialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1626   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1627   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1628   beforestart=0,language=2}

```

**\babelensure** The user command just parses the optional argument and creates a new macro named `\bbl@e@<language>`. We register a hook at the `afterextras` event which just executes this macro in a “complete” selection (which, if undefined, is `\relax` and does nothing). This part is somewhat involved because we have to make sure things are expanded the correct number of times.

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

```

1629 \bbl@trace{Defining babelensure}
1630 \newcommand\babelensure[2][{}]{% TODO - revise test files
1631   \AddBabelHook{babel-ensure}{afterextras}{%
1632     \ifcase\bbl@select@type
1633       \bbl@cl{e}%
1634     \fi}%
1635   \begingroup
1636     \let\bbl@ens@include\@empty
1637     \let\bbl@ens@exclude\@empty
1638     \def\bbl@ens@fontenc{\relax}%
1639     \def\bbl@tempb##1{%
1640       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1641     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1642     \def\bbl@tempb##1=##2\@{\@namedef{bbl@ens@##1}{##2}}%
1643     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1644     \def\bbl@tempc{\bbl@ensure}%

```

```

1645 \expandafter\bb1@add\expandafter\bb1@tempc\expandafter{%
1646 \expandafter{\bb1@ens@include}}%
1647 \expandafter\bb1@add\expandafter\bb1@tempc\expandafter{%
1648 \expandafter{\bb1@ens@exclude}}%
1649 \toks@\expandafter{\bb1@tempc}%
1650 \bb1@exp{%
1651 \endgroup
1652 \def\<bb1@e@#2>{\the\toks@{\bb1@ens@fontenc}}}%
1653 \def\bb1@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1654 \def\bb1@tempb##1{% elt for (excluding) \bb1@captionslist list
1655 \ifx##1\undefined % 3.32 - Don't assume the macro exists
1656 \edef##1{\noexpand\bb1@nocaption
1657 {\bb1@stripslash##1}{\language\bb1@stripslash##1}}%
1658 \fi
1659 \ifx##1\@empty\else
1660 \in@{##1}{#2}%
1661 \ifin@ \else
1662 \bb1@ifunset{\bb1@ensure@\language}%
1663 {\bb1@exp{%
1664 \\\DeclareRobustCommand\<bb1@ensure@\language>[1]{%
1665 \\\foreignlanguage{\language}%
1666 {\ifx\relax#3\else
1667 \\\fontencoding{#3}\selectfont
1668 \fi
1669 #####1}}}%
1670 {}%
1671 \toks@\expandafter{##1}%
1672 \edef##1{%
1673 \bb1@csarg\noexpand{ensure@\language}%
1674 {\the\toks@}}%
1675 \fi
1676 \expandafter\bb1@tempb
1677 \fi}%
1678 \expandafter\bb1@tempb\bb1@captionslist\today\@empty
1679 \def\bb1@tempa##1{% elt for include list
1680 \ifx##1\@empty\else
1681 \bb1@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
1682 \ifin@ \else
1683 \bb1@tempb##1\@empty
1684 \fi
1685 \expandafter\bb1@tempa
1686 \fi}%
1687 \bb1@tempa#1\@empty}
1688 \def\bb1@captionslist{%
1689 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1690 \contentsname\listfigurename\listtablename\indexname\figurename
1691 \tablename\partname\enc1name\ccname\headtoname\pagename\seename
1692 \alsoname\proofname\glossaryname}

```

## 9.4 Setting up language files

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

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

Another character that needs to have the correct category code during processing of language definition files is the equals sign, ‘=’, because it is sometimes used in constructions with the `\let` primitive. Therefore we store its current catcode and restore it later on. Now we check whether we should perhaps stop the processing of this file. To do this we first need to check whether the second argument that is passed to `\LdfInit` is a control sequence. We do that by looking at the first token after passing #2 through string. When it is equal to `\@backslashchar` we are dealing with a control sequence which we can compare with `\@undefined`.

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

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

Finally we check `\originalTeX`.

```

1693 \bbl@trace{Macros for setting language files up}
1694 \def\bbl@ldfinit{% TODO. Merge into the next macro? Unused elsewhere
1695   \let\bbl@screset\@empty
1696   \let\BabelStrings\bbl@opt@string
1697   \let\BabelOptions\@empty
1698   \let\BabelLanguages\relax
1699   \ifx\originalTeX\@undefined
1700     \let\originalTeX\@empty
1701   \else
1702     \originalTeX
1703   \fi}
1704 \def\LdfInit#1#2{%
1705   \chardef\atcatcode=\catcode`\@
1706   \catcode`\@=11\relax
1707   \chardef\eqcatcode=\catcode`\=
1708   \catcode`\==12\relax
1709   \expandafter\if\expandafter\@backslashchar
1710     \expandafter\@car\string#2\@nil
1711   \ifx#2\@undefined\else
1712     \ldf@quit{#1}%
1713   \fi
1714 \else
1715   \expandafter\ifx\csname#2\endcsname\relax\else
1716     \ldf@quit{#1}%
1717   \fi
1718 \fi
1719 \bbl@ldfinit}

```

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

```

1720 \def\ldf@quit#1{%
1721   \expandafter\main@language\expandafter{#1}%
1722   \catcode`\@=\atcatcode \let\atcatcode\relax
1723   \catcode`\==\eqcatcode \let\eqcatcode\relax
1724   \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.

```

1725 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1726   \bbl@afterlang
1727   \let\bbl@afterlang\relax
1728   \let\BabelModifiers\relax

```

```

1729 \let\bbl@screset\relax}%
1730 \def\ldf@finish#1{%
1731 \ifx\loadlocalcfg\undefined\else % For LaTeX 209
1732 \loadlocalcfg{#1}%
1733 \fi
1734 \bbl@afterldf{#1}%
1735 \expandafter\main@language\expandafter{#1}%
1736 \catcode`\@=\atcatcode \let\atcatcode\relax
1737 \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  $\LaTeX$ .

```

1738 \@onlypreamble\LdfInit
1739 \@onlypreamble\ldf@quit
1740 \@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.

```

1741 \def\main@language#1{%
1742 \def\bbl@main@language{#1}%
1743 \let\languagename\bbl@main@language % TODO. Set localename
1744 \bbl@id@assign
1745 \bbl@patterns{\languagename}}

```

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

```

1746 \def\bbl@beforestart{%
1747 \bbl@usehooks{beforestart}{}}%
1748 \global\let\bbl@beforestart\relax}
1749 \AtBeginDocument{%
1750 \@nameuse{bbl@beforestart}%
1751 \if@filesw
1752 \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}}%
1753 \fi
1754 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1755 \ifbbl@single % must go after the line above.
1756 \renewcommand\selectlanguage[1]{}%
1757 \renewcommand\foreignlanguage[2]{#2}%
1758 \global\let\babel@aux\@gobbletwo % Also as flag
1759 \fi
1760 \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1761 \def\select@language@x#1{%
1762 \ifcase\bbl@select@type
1763 \bbl@ifsamestring\languagename{#1}{\select@language{#1}}%
1764 \else
1765 \select@language{#1}%
1766 \fi}

```

## 9.5 Shorthands

`\bbl@add@special` The macro `\bbl@add@special` is used to add a new character (or single character control sequence) to the macro `\dospecials` (and `\@sanitize` if  $\LaTeX$  is used). It is used only at

one place, namely when `\initiate@active@char` is called (which is ignored if the char has been made active before). Because `\@sanitize` can be undefined, we put the definition inside a conditional.

Items are added to the lists without checking its existence or the original catcode. It does not hurt, but should be fixed. It's already done with `\nfss@catcodes`, added in 3.10.

```

1767 \bbl@trace{Shorhands}
1768 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1769   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1770   \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\@makeother#1}}%
1771   \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1772     \begingroup
1773       \catcode`#1\active
1774       \nfss@catcodes
1775       \ifnum\catcode`#1=\active
1776         \endgroup
1777         \bbl@add\nfss@catcodes{\@makeother#1}%
1778       \else
1779         \endgroup
1780       \fi
1781   \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.

```

1782 \def\bbl@remove@special#1{%
1783   \begingroup
1784   \def\x##1##2{\ifnum`#1=##2\noexpand\@empty
1785     \else\noexpand##1\noexpand##2\fi}%
1786   \def\do{\x\do}%
1787   \def\@makeother{\x\@makeother}%
1788   \edef\x{\endgroup
1789     \def\noexpand\dospecials{\dospecials}%
1790     \expandafter\ifx\csname @sanitize\endcsname\relax\else
1791       \def\noexpand\@sanitize{\@sanitize}%
1792     \fi}%
1793   \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).



```

1794 \def\bbl@active@def#1#2#3#4{%
1795   \namedef{#3#1}{%
1796     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1797       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1798     \else
1799       \bbl@afterfi\csname#2@sh@#1\endcsname
1800     \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.

```

1801 \long\@namedef{#3@arg#1}##1{%
1802   \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
1803     \bbl@afterelse\csname#4#1\endcsname##1%
1804   \else
1805     \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
1806   \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.

```

1807 \def\@initiate@active@char#1{%
1808   \bbl@ifunset{active@char\string#1}%
1809   {\bbl@withactive
1810     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1811   {}}

```

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).

```

1812 \def\@initiate@active@char#1#2#3{%
1813   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1814   \ifx#1\@undefined
1815     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
1816   \else
1817     \bbl@csarg\let{oridef@#2}#1%
1818     \bbl@csarg\edef{oridef@#2}{%
1819       \let\noexpand#1%
1820       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1821   \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*").

```

1822 \ifx#1#3\relax
1823   \expandafter\let\csname normal@char#2\endcsname#3%
1824 \else
1825   \bbl@info{Making #2 an active character}%
1826   \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1827   \@namedef{normal@char#2}{%
1828     \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1829   \else
1830     \@namedef{normal@char#2}{#3}%
1831   \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).

```

1832 \bbl@restoreactive{#2}%
1833 \AtBeginDocument{%
1834   \catcode`#2\active
1835   \if@filesw
1836     \immediate\write\@mainaux{\catcode`\string#2\active}%
1837   \fi}%
1838 \expandafter\bbl@add@special\csname#2\endcsname
1839 \catcode`#2\active
1840 \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⟩`).

```

1841 \let\bbl@tempa\@firstoftwo
1842 \if\string^#2%
1843   \def\bbl@tempa{\noexpand\textormath}%
1844 \else
1845   \ifx\bbl@mathnormal\@undefined\else
1846     \let\bbl@tempa\bbl@mathnormal
1847   \fi
1848 \fi
1849 \expandafter\edef\csname active@char#2\endcsname{%
1850   \bbl@tempa
1851   {\noexpand\if@safe@actives
1852     \noexpand\expandafter
1853     \expandafter\noexpand\csname normal@char#2\endcsname
1854   \noexpand\else
1855     \noexpand\expandafter
1856     \expandafter\noexpand\csname bbl@doactive#2\endcsname
1857   \noexpand\fi}}%
1858 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1859 \bbl@csarg\edef{doactive#2}{%
1860   \expandafter\noexpand\csname user@active#2\endcsname}%

```

We now define the default values which the shorthand is set to when activated or deactivated. It is set to the deactivated form (globally), so that the character expands to

`\active@prefix⟨char⟩\normal@char⟨char⟩`

(where `\active@char⟨char⟩` is *one* control sequence!).

```

1861 \bbl@csarg\edef{active@#2}{%
1862   \noexpand\active@prefix\noexpand#1%
1863   \expandafter\noexpand\csname active@char#2\endcsname}%
1864 \bbl@csarg\edef{normal@#2}{%
1865   \noexpand\active@prefix\noexpand#1%
1866   \expandafter\noexpand\csname normal@char#2\endcsname}%
1867 \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.

```

1868 \bbl@active@def#2\user@group{user@active}{language@active}%
1869 \bbl@active@def#2\language@group{language@active}{system@active}%
1870 \bbl@active@def#2\system@group{system@active}{normal@char}%

```

In order to do the right thing when a shorthand with an argument is used by itself at the end of the line we provide a definition for the case of an empty argument. For that case we let the shorthand character expand to its non-active self. Also, When a shorthand combination such as ' ' ends up in a heading  $\TeX$  would see `\protect'\protect'`. To prevent this from happening a couple of shorthand needs to be defined at user level.

```

1871 \expandafter\edef\csname\user@group @sh@#2@@\endcsname
1872 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1873 \expandafter\edef\csname\user@group @sh@#2@\string\protect\endcsname
1874 {\expandafter\noexpand\csname user@active#2\endcsname}%

```

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

```

1875 \if\string'#2%
1876 \let\prim@s\bbl@prim@s
1877 \let\active@math@prime#1%
1878 \fi
1879 \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}

```

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

```

1880 <<(*More package options)>> ≡
1881 \DeclareOption{math=active}{}
1882 \DeclareOption{math=normal}{{\def\bbl@mathnormal{\noexpand\textormath}}}
1883 <</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*.

```

1884 \@ifpackagewith{babel}{KeepShorthandsActive}%
1885 {\let\bbl@restoreactive@gobble}%
1886 {\def\bbl@restoreactive#1{%
1887   \bbl@exp{%
1888     \\\AfterBabelLanguage\\CurrentOption
1889     {\catcode`#1=\the\catcode`#1\relax}%
1890     \\\AtEndOfPackage
1891     {\catcode`#1=\the\catcode`#1\relax}}}%
1892 \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.

```

1893 \def\bbl@sh@select#1#2{%
1894 \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1895   \bbl@afterelse\bbl@scndcs
1896 \else
1897   \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1898 \fi}

```

`\active@prefix` The command `\active@prefix` which is used in the expansion of active characters has a function similar to `\OT1-cmd` in that it `\protects` the active character whenever `\protect` is *not* `\@typeset@protect`. The `\@gobble` is needed to remove a token such as `\activechar:` (when the double colon was the active character to be dealt with). There are two definitions, depending of `\ifincsname` is available. If there is, the expansion will be more robust.

```

1899 \begingroup
1900 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
1901   {\gdef\active@prefix#1{%
1902     \ifx\protect\@typeset@protect
1903       \else
1904         \ifx\protect\@unexpandable@protect
1905           \noexpand#1%
1906         \else
1907           \protect#1%
1908         \fi
1909         \expandafter\@gobble
1910       \fi}}
1911   {\gdef\active@prefix#1{%
1912     \ifincsname
1913       \string#1%
1914       \expandafter\@gobble
1915     \else
1916       \ifx\protect\@typeset@protect
1917         \else
1918           \ifx\protect\@unexpandable@protect
1919             \noexpand#1%
1920           \else
1921             \protect#1%
1922           \fi
1923           \expandafter\expandafter\expandafter\@gobble
1924         \fi
1925       \fi}}
1926 \endgroup

```

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

```

1927 \newif\if@safe@actives
1928 \@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.

```

1929 \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  
`\bbl@deactivate` 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`.

```

1930 \def\bbl@activate#1{%
1931   \bbl@withactive{\expandafter\let\expandafter}#1%
1932   \csname bbl@active@\string#1\endcsname}
1933 \def\bbl@deactivate#1{%
1934   \bbl@withactive{\expandafter\let\expandafter}#1%
1935   \csname bbl@normal@\string#1\endcsname}

```

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

`\bbl@scndcs` 1936 `\def\bbl@firstcs#1#2{\csname#1\endcsname}`  
 1937 `\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.

```

1938 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1939 \def\@decl@short#1#2#3\@nil#4{%
1940   \def\bbl@tempa{#3}%
1941   \ifx\bbl@tempa\@empty
1942     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1943     \bbl@ifunset{#1@sh@\string#2@}\{%
1944       {\def\bbl@tempa{#4}%
1945         \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1946         \else
1947           \bbl@info
1948             {Redefining #1 shorthand \string#2\\
1949              in language \CurrentOption}%
1950           \fi}%
1951       \@namedef{#1@sh@\string#2@}{#4}%
1952     \else
1953       \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1954       \bbl@ifunset{#1@sh@\string#2@\string#3@}\{%
1955         {\def\bbl@tempa{#4}%
1956         \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1957         \else
1958           \bbl@info
1959             {Redefining #1 shorthand \string#2\string#3\\
1960              in language \CurrentOption}%
1961           \fi}%
1962         \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1963       \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.

```

1964 \def\textormath{%
1965   \ifmmode
1966     \expandafter\@secondoftwo
1967   \else
1968     \expandafter\@firstoftwo
1969   \fi}

```

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

`\language@group`

`\system@group`

```

1970 \def\user@group{user}
1971 \def\language@group{english} % TODO. I don't like defaults
1972 \def\system@group{system}

```

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

```

1973 \def\useshortands{%
1974   \@ifstar\bb@usesh@s{\bb@usesh@x{}}
1975 \def\bb@usesh@s#1{%
1976   \bb@usesh@x
1977   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb@activate{#1}}}%
1978   {#1}}
1979 \def\bb@usesh@x#1#2{%
1980   \bb@ifshorthand{#2}%
1981   {\def\user@group{user}%
1982    \initiate@active@char{#2}%
1983    #1%
1984    \bb@activate{#2}}%
1985   {\bb@error
1986    {Cannot declare a shorthand turned off (\string#2)}
1987    {Sorry, but you cannot use shortcuts which have been\%
1988     turned off in the package options}}}

```

`\defineshorthand` Currently we only support two groups of user level shortcuts, named internally user and user@<lang> (language-dependent user shortcuts). 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 `\bb@set@user@generic`); we make also sure {} and \protect are taken into account in this new top level.

```

1989 \def\user@language@group{user@\language@group}
1990 \def\bb@set@user@generic#1#2{%
1991   \bb@ifunset{user@generic@active#1}%
1992   {\bb@active@def#1\user@language@group{user@active}{user@generic@active}%
1993    \bb@active@def#1\user@group{user@generic@active}{language@active}%
1994    \expandafter\edef\csname#2sh@#1@%\endcsname{
1995      \expandafter\noexpand\csname normal@char#1\endcsname}%
1996    \expandafter\edef\csname#2sh@#1@\string\protect%\endcsname{
1997      \expandafter\noexpand\csname user@active#1\endcsname}}%
1998   \@empty}
1999 \newcommand\defineshorthand[3][user]{%
2000   \edef\bb@tempa{\zap@space#1 \@empty}%
2001   \bb@for\bb@tempb\bb@tempa{%
2002     \if*\expandafter\@car\bb@tempb\@nil
2003       \edef\bb@tempb{user@\expandafter\@gobble\bb@tempb}%
2004       \@expandtwoargs
2005       \bb@set@user@generic{\expandafter\string\@car#2\@nil}\bb@tempb
2006     \fi
2007     \declare@shorthand{\bb@tempb}{#2}{#3}}

```

`\languageshortands` A user level command to change the language from which shortcuts 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 [TODO. Unclear].

```

2008 \def\languageshortands#1{\def\language@group{#1}}

```

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

```

2009 \def\aliasshorthand#1#2{%
2010   \bb@ifshorthand{#2}%
2011   {\expandafter\ifx\csname active@char\string#2\endcsname\relax

```

```

2012 \ifx\document\@notprerr
2013 \@notshorthand{#2}%
2014 \else
2015 \initiate@active@char{#2}%
2016 \expandafter\let\csname active@char\string#2\expandafter\endcsname
2017 \csname active@char\string#1\endcsname
2018 \expandafter\let\csname normal@char\string#2\expandafter\endcsname
2019 \csname normal@char\string#1\endcsname
2020 \bbl@activate{#2}%
2021 \fi
2022 \fi}%
2023 {\bbl@error
2024 {Cannot declare a shorthand turned off (\string#2)}
2025 {Sorry, but you cannot use shorthands which have been\\%
2026 turned off in the package options}}}

```

\@notshorthand

```

2027 \def\@notshorthand#1{%
2028 \bbl@error{%
2029 The character '\string #1' should be made a shorthand character;\\%
2030 add the command \string\usesshorthands\string{#1\string} to
2031 the preamble.\\%
2032 I will ignore your instruction}%
2033 {You may proceed, but expect unexpected results}}

```

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

```

2034 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2035 \DeclareRobustCommand*\shorthandoff{%
2036 \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2037 \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.

```

2038 \def\bbl@switch@sh#1#2{%
2039 \ifx#2\@nnil\else
2040 \bbl@ifunset{\bbl@active@\string#2}%
2041 {\bbl@error
2042 {I cannot switch '\string#2' on or off--not a shorthand}%
2043 {This character is not a shorthand. Maybe you made\\%
2044 a typing mistake? I will ignore your instruction}}}%
2045 {\ifcase#1%
2046 \catcode`#212\relax
2047 \or
2048 \catcode`#2\active
2049 \or
2050 \csname bbl@oricat@\string#2\endcsname
2051 \csname bbl@oridef@\string#2\endcsname
2052 \fi}%
2053 \bbl@afterfi\bbl@switch@sh#1%
2054 \fi}

```

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

```

2055 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
2056 \def\bbl@putsh#1{%
2057   \bbl@ifunset{\bbl@active@\string#1}%
2058   {\bbl@putsh@i#1\@empty\@nnil}%
2059   {\csname bbl@active@\string#1\endcsname}}
2060 \def\bbl@putsh@i#1#2\@nnil{%
2061   \csname\language\@sh@\string#1@%
2062     \ifx\@empty#2\else\string#2\fi\endcsname}
2063 \ifx\bbl@opt@shorthands\@nnil\else
2064   \let\bbl@s@initiate@active@char\initiate@active@char
2065   \def\initiate@active@char#1{%
2066     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
2067   \let\bbl@s@switch@sh\bbl@switch@sh
2068   \def\bbl@switch@sh#1#2{%
2069     \ifx#2\@nnil\else
2070       \bbl@afterfi
2071       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
2072       \fi}
2073   \let\bbl@s@activate\bbl@activate
2074   \def\bbl@activate#1{%
2075     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
2076   \let\bbl@s@deactivate\bbl@deactivate
2077   \def\bbl@deactivate#1{%
2078     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
2079 \fi

```

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

```

2080 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{\bbl@active@\string#1}{#3}{#2}}

```

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

**\bbl@pr@m@s**

```

2081 \def\bbl@prim@s{%
2082   \prime\futurelet\@let@token\bbl@pr@m@s}
2083 \def\bbl@if@primes#1#2{%
2084   \ifx#1\@let@token
2085     \expandafter\@firstoftwo
2086   \else\ifx#2\@let@token
2087     \bbl@afterelse\expandafter\@firstoftwo
2088   \else
2089     \bbl@afterfi\expandafter\@secondoftwo
2090   \fi\fi}
2091 \begingroup
2092 \catcode`\^=7 \catcode`\*= \active \lccode`\^=\^
2093 \catcode`\'=12 \catcode`\="= \active \lccode`\"="\ '
2094 \lowercase{%
2095   \gdef\bbl@pr@m@s{%
2096     \bbl@if@primes"%
2097       \pr@@@s
2098       {\bbl@if@primes*\^{\pr@@@t\egroup}}}
2099 \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).

```
2100 \initiate@active@char{~}
2101 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2102 \bbl@activate{~}
```

\OT1dpos The position of the double quote character is different for the OT1 and T1 encodings. It will later be selected using the \f@encoding macro. Therefore we define two macros here to store the position of the character in these encodings.

```
2103 \expandafter\def\csname OT1dpos\endcsname{127}
2104 \expandafter\def\csname T1dpos\endcsname{4}
```

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

```
2105 \ifx\f@encoding\undefined
2106   \def\f@encoding{OT1}
2107 \fi
```

## 9.6 Language attributes

Language attributes provide a means to give the user control over which features of the language definition files he wants to enable.

\languageattribute The macro \languageattribute checks whether its arguments are valid and then activates the selected language attribute. First check whether the language is known, and then process each attribute in the list.

```
2108 \bbl@trace{Language attributes}
2109 \newcommand\languageattribute[2]{%
2110   \def\bbl@tempc{#1}%
2111   \bbl@fixname\bbl@tempc
2112   \bbl@iflanguage\bbl@tempc{%
2113     \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.

```
2114     \ifx\bbl@known@attribs\undefined
2115       \in@false
2116     \else
2117       \bbl@xin@{,\bbl@tempc-##1,},{,\bbl@known@attribs,}%
2118     \fi
2119     \ifin@
2120       \bbl@warning{%
2121         You have more than once selected the attribute '##1'\%
2122         for language #1. Reported}%
2123     \else
```

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

```
2124     \bbl@exp{%
2125       \\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
2126     \edef\bbl@tempa{\bbl@tempc-##1}%
2127     \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
2128     {\csname\bbl@tempc @attr##1\endcsname}%
2129     {\@attrerr{\bbl@tempc}{##1}}%
```

```

2130     \fi}}
2131 \@onlypreamble\languageattribute

```

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

```

2132 \newcommand*{\@attrerr}[2]{%
2133   \bbl@error
2134   {The attribute #2 is unknown for language #1.}%
2135   {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}`.

```

2136 \def\bbl@declare@ttribute#1#2#3{%
2137   \bbl@xin@{,#2,},{,\BabelModifiers,}%
2138   \ifin@
2139     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2140   \fi
2141   \bbl@add@list\bbl@attributes{#1-#2}%
2142   \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.  
First we need to find out if any attributes were set; if not we're done. Then we need to check the list of known attributes. 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`'.

```

2143 \def\bbl@ifattributeset#1#2#3#4{%
2144   \ifx\bbl@known@attribs\@undefined
2145     \in@false
2146   \else
2147     \bbl@xin@{,#1-#2,},{,\bbl@known@attribs,}%
2148   \fi
2149   \ifin@
2150     \bbl@afterelse#3%
2151   \else
2152     \bbl@afterfi#4%
2153   \fi
2154 }

```

`\bbl@ifknown@trib` An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the  $\TeX$ -code to be executed when the attribute is known and the  $\TeX$ -code to be executed otherwise.  
We first assume the attribute is unknown. Then we loop over the list of known attributes, trying to find a match. When a match is found the definition of `\bbl@tempa` is changed. Finally we execute `\bbl@tempa`.

```

2155 \def\bbl@ifknown@trib#1#2{%
2156   \let\bbl@tempa\@secondoftwo
2157   \bbl@loopx\bbl@tempb{#2}{%
2158     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%

```

```

2159 \ifin@
2160 \let\bbl@tempa\@firstoftwo
2161 \else
2162 \fi}%
2163 \bbl@tempa
2164 }

```

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

```

2165 \def\bbl@clear@ttribs{%
2166 \ifx\bbl@attributes\@undefined\else
2167 \bbl@loopx\bbl@tempa{\bbl@attributes}{%
2168 \expandafter\bbl@clear@ttrib\bbl@tempa.
2169 }%
2170 \let\bbl@attributes\@undefined
2171 \fi}
2172 \def\bbl@clear@ttrib#1-#2.{%
2173 \expandafter\let\csname#1@attr#2\endcsname\@undefined}
2174 \AtBeginDocument{\bbl@clear@ttribs}

```

## 9.7 Support for saving macro definitions

To save the meaning of control sequences using `\babel@save`, we use temporary control sequences. To save hash table entries for these control sequences, we don't use the name of the control sequence to be saved to construct the temporary name. Instead we simply use the value of a counter, which is reset to zero each time we begin to save new values. This works well because we release the saved meanings before we begin to save a new set of control sequence meanings (see `\selectlanguage` and `\originalTeX`). Note undefined macros are not undefined any more when saved – they are `\relax`'ed.

`\babel@savecnt` The initialization of a new save cycle: reset the counter to zero.

`\babel@beginsave`

```

2175 \bbl@trace{Macros for saving definitions}
2176 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

2177 \newcount\babel@savecnt
2178 \babel@beginsave

```

`\babel@save` The macro `\babel@save<cname>` saves the current meaning of the control sequence `<cname>` to `\originalTeX`<sup>31</sup>. To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to `\originalTeX` and the counter is incremented. The macro `\babel@savevariable<variable>` saves the value of the variable. `<variable>` can be anything allowed after the `\the` primitive.

```

2179 \def\babel@save#1{%
2180 \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
2181 \toks@\expandafter{\originalTeX\let#1=}
2182 \bbl@exp{%
2183 \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
2184 \advance\babel@savecnt\@ne}
2185 \def\babel@savevariable#1{%
2186 \toks@\expandafter{\originalTeX #1=}
2187 \bbl@exp{\def\\originalTeX{\the\toks@<\the#1>\relax}}

```

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

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

```
2188 \def\bbl@frenchspacing{%
2189   \ifnum\the\sfcode`\.=\@m
2190     \let\bbl@nonfrenchspacing\relax
2191   \else
2192     \frenchspacing
2193     \let\bbl@nonfrenchspacing\nonfrenchspacing
2194   \fi}
2195 \let\bbl@nonfrenchspacing\nonfrenchspacing
```

## 9.8 Short tags

`\babeltags` This macro is straightforward. After zapping spaces, we loop over the list and define the macros `\text{<tag>}` and `\<tag>`. Definitions are first expanded so that they don't contain `\csname` but the actual macro.

```
2196 \bbl@trace{Short tags}
2197 \def\babeltags#1{%
2198   \edef\bbl@tempa{\zap@space#1 \@empty}%
2199   \def\bbl@tempb##1=##2\@{#}%
2200   \edef\bbl@tempc{%
2201     \noexpand\newcommand
2202     \expandafter\noexpand\csname ##1\endcsname{%
2203       \noexpand\protect
2204       \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
2205     \noexpand\newcommand
2206     \expandafter\noexpand\csname text##1\endcsname{%
2207       \noexpand\foreignlanguage{##2}}
2208     \bbl@tempc}%
2209   \bbl@for\bbl@tempa\bbl@tempa{%
2210     \expandafter\bbl@tempb\bbl@tempa\@{}}
```

## 9.9 Hyphens

`\babelhyphenation` This macro saves hyphenation exceptions. Two macros are used to store them: `\bbl@hyphenation@` for the global ones and `\bbl@hyphenation<lang>` for language ones. See `\bbl@patterns` above for further details. We make sure there is a space between words when multiple commands are used.

```
2211 \bbl@trace{Hyphens}
2212 \@onlypreamble\babelhyphenation
2213 \AtEndOfPackage{%
2214   \newcommand\babelhyphenation[2][\@empty]{%
2215     \ifx\bbl@hyphenation@\relax
2216       \let\bbl@hyphenation@\@empty
2217     \fi
2218     \ifx\bbl@hyphlist\@empty\else
2219       \bbl@warning{%
2220         You must not intermingle \string\selectlanguage\space and\%
2221         \string\babelhyphenation\space or some exceptions will not\%
2222         be taken into account. Reported}%
2223     \fi
2224     \ifx\@empty#1%
2225       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2226     \else
2227       \bbl@vforeach{#1}{%
```

```

2228 \def\bb@tempa{##1}%
2229 \bb@fixname\bb@tempa
2230 \bb@iflanguage\bb@tempa{%
2231 \bb@csarg\protected@edef{hyphenation@\bb@tempa}{%
2232 \bb@ifunset{bb@hyphenation@\bb@tempa}%
2233 \@empty
2234 {\csname bb@hyphenation@\bb@tempa\endcsname\space}%
2235 #2}}}%
2236 \fi}}

```

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

```

2237 \def\bb@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
2238 \def\bb@t@one{T1}
2239 \def\allowhyphens{\ifx\cf@encoding\bb@t@one\else\bb@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`.

```

2240 \newcommand\babelnullhyphen{\char\hyphenchar\font}
2241 \def\babelhyphen{\active@prefix\babelhyphen\bb@hyphen}
2242 \def\bb@hyphen{%
2243 \@ifstar{\bb@hyphen@i @}{\bb@hyphen@i \@empty}}
2244 \def\bb@hyphen@i#1#2{%
2245 \bb@ifunset{bb@hy@#1#2\@empty}%
2246 {\csname bb@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
2247 {\csname bb@hy@#1#2\@empty\endcsname}}

```

The following two commands are used to wrap the “hyphen” and set the behavior of the rest of the word – the version with a single @ is used when further hyphenation is allowed, while that with @@ if no more hyphens are allowed. In both cases, if the hyphen is preceded by a positive space, breaking after the hyphen is disallowed. There should not be a discretionary after a hyphen at the beginning of a word, so it is prevented if preceded by a skip. Unfortunately, this does handle cases like “(-suffix)”. `\nobreak` is always preceded by `\leavevmode`, in case the shorthand starts a paragraph.

```

2248 \def\bb@usehyphen#1{%
2249 \leavevmode
2250 \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2251 \nobreak\hskip\z@skip}
2252 \def\bb@@usehyphen#1{%
2253 \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

2254 \def\bb@hyphenchar{%
2255 \ifnum\hyphenchar\font=\m@ne
2256 \babelnullhyphen
2257 \else
2258 \char\hyphenchar\font
2259 \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 `\bb@hy@nobreak` is redundant.

```

2260 \def\bb@hy@soft{\bb@usehyphen{\discretionary{\bb@hyphenchar}{}}{}}
2261 \def\bb@hy@@soft{\bb@usehyphen{\discretionary{\bb@hyphenchar}{}}{}}
2262 \def\bb@hy@hard{\bb@usehyphen\bb@hyphenchar}
2263 \def\bb@hy@@hard{\bb@usehyphen\bb@hyphenchar}

```

<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.

```

2264 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
2265 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
2266 \def\bbl@hy@repeat{%
2267   \bbl@usehyphen{%
2268     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
2269 \def\bbl@hy@repeat{%
2270   \bbl@usehyphen{%
2271     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
2272 \def\bbl@hy@empty{\hskip\z@skip}
2273 \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.

```

2274 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#1}\bbl@allowhyphens}

```

## 9.10 Multiencoding strings

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

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

```

2275 \bbl@trace{Multiencoding strings}
2276 \def\bbl@tglobal#1{\global\let#1#1}
2277 \def\bbl@recatcode#1{% TODO. Used only once?
2278   \@tempcnta="7F
2279   \def\bbl@tempa{%
2280     \ifnum\@tempcnta>"FF\else
2281       \catcode\@tempcnta=#1\relax
2282       \advance\@tempcnta\@ne
2283       \expandafter\bbl@tempa
2284     \fi}%
2285   \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).

```

2286 \@ifpackagewith{babel}{nocase}%
2287 {\let\bbl@patchuclc\relax}%
2288 {\def\bbl@patchuclc{%
2289   \global\let\bbl@patchuclc\relax
2290   \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
2291   \gdef\bbl@uclc##1{%
2292     \let\bbl@encoded\bbl@encoded@uclc
2293     \bbl@ifunset{\language @bbl@uclc}% and resumes it
2294     {##1}%
2295     {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc

```

```

2296 \csname\language @bbl@uc\endcsname}%
2297 {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
2298 \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
2299 \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}%
2300 <<(*More package options)>> ≡
2301 \DeclareOption{nocase}{}
2302 <</More package options>>

```

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

```

2303 <<(*More package options)>> ≡
2304 \let\bbl@opt@strings\@nnil % accept strings=value
2305 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2306 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2307 \def\BabelStringsDefault{generic}
2308 <</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.

```

2309 \@onlypreamble\StartBabelCommands
2310 \def\StartBabelCommands{%
2311   \begingroup
2312   \bbl@recatcode{11}%
2313   <<Macros local to BabelCommands>>
2314   \def\bbl@provstring##1##2{%
2315     \providecommand##1{##2}%
2316     \bbl@tglobal##1}%
2317   \global\let\bbl@scafter\@empty
2318   \let\StartBabelCommands\bbl@startcmds
2319   \ifx\BabelLanguages\relax
2320     \let\BabelLanguages\CurrentOption
2321   \fi
2322   \begingroup
2323   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2324   \StartBabelCommands}
2325 \def\bbl@startcmds{%
2326   \ifx\bbl@screset\@nnil\else
2327     \bbl@usehooks{stopcommands}{}%
2328   \fi
2329   \endgroup
2330   \begingroup
2331   \@ifstar
2332     {\ifx\bbl@opt@strings\@nnil
2333       \let\bbl@opt@strings\BabelStringsDefault
2334       \fi
2335       \bbl@startcmds@i}%
2336     \bbl@startcmds@i}
2337 \def\bbl@startcmds@i#1#2{%
2338   \edef\bbl@L{\zap@space#1 \@empty}%
2339   \edef\bbl@G{\zap@space#2 \@empty}%
2340   \bbl@startcmds@ii}
2341 \let\bbl@startcmds\StartBabelCommands

```

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

Select the behavior of `\SetString`. There are two main cases, depending of if there is an optional argument: without it and `strings=encoded`, strings are defined always; otherwise, they are set only if they are still undefined (ie, fallback values). With labelled blocks and `strings=encoded`, define the strings, but with another value, define strings

only if the current label or font encoding is the value of strings; otherwise (ie, no strings or a block whose label is not in strings=) do nothing.

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

```

2342 \newcommand\bb1@startcmds@ii[1][\@empty]{%
2343   \let\SetString\@gobbletwo
2344   \let\bb1@stringdef\@gobbletwo
2345   \let\AfterBabelCommands\@gobble
2346   \ifx\@empty#1%
2347     \def\bb1@sc@label{generic}%
2348     \def\bb1@encstring##1##2{%
2349       \ProvideTextCommandDefault##1{##2}%
2350       \bb1@tglobal##1%
2351       \expandafter\bb1@tglobal\csname\string?\string##1\endcsname}%
2352     \let\bb1@sctest\in@true
2353   \else
2354     \let\bb1@sc@charset\space % <- zapped below
2355     \let\bb1@sc@fontenc\space % <- " "
2356     \def\bb1@tempa##1=##2\@nil{%
2357       \bb1@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
2358     \bb1@foreach{label=#1}{\bb1@tempa##1\@nil}%
2359     \def\bb1@tempa##1 ##2{% space -> comma
2360       ##1%
2361       \ifx\@empty##2\else\ifx,##1,\else,\fi\bb1@afterfi\bb1@tempa##2\fi}%
2362     \edef\bb1@sc@fontenc{\expandafter\bb1@tempa\bb1@sc@fontenc\@empty}%
2363     \edef\bb1@sc@label{\expandafter\zap@space\bb1@sc@label\@empty}%
2364     \edef\bb1@sc@charset{\expandafter\zap@space\bb1@sc@charset\@empty}%
2365     \def\bb1@encstring##1##2{%
2366       \bb1@foreach\bb1@sc@fontenc{%
2367         \bb1@ifunset{T####1}%
2368         {}%
2369         {\ProvideTextCommand##1{####1}{##2}%
2370         \bb1@tglobal##1%
2371         \expandafter
2372         \bb1@tglobal\csname####1\string##1\endcsname}}}%
2373     \def\bb1@sctest{%
2374       \bb1@xin@{\bb1@opt@strings,}{,\bb1@sc@label,\bb1@sc@fontenc,}}%
2375   \fi
2376   \ifx\bb1@opt@strings\@nnil % ie, no strings key -> defaults
2377   \else\ifx\bb1@opt@strings\relax % ie, strings=encoded
2378     \let\AfterBabelCommands\bb1@aftercmds
2379     \let\SetString\bb1@setstring
2380     \let\bb1@stringdef\bb1@encstring
2381   \else % ie, strings=value
2382     \bb1@sctest
2383   \fin@
2384     \let\AfterBabelCommands\bb1@aftercmds
2385     \let\SetString\bb1@setstring
2386     \let\bb1@stringdef\bb1@provstring
2387   \fi\fi\fi
2388   \bb1@scswitch
2389   \ifx\bb1@G\@empty
2390     \def\SetString##1##2{%
2391       \bb1@error{Missing group for string \string##1}%
2392       {You must assign strings to some category, typically\\%
2393       captions or extras, but you set none}}%
2394   \fi

```



```

2395 \ifx\@empty#1%
2396 \bbl@usehooks{defaultcommands}{}%
2397 \else
2398 \@expandtwoargs
2399 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
2400 \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) .

```

2401 \def\bbl@forlang#1#2{%
2402 \bbl@for#1\bbl@L{%
2403 \bbl@xin@{,#1,}{,\BabelLanguages,}%
2404 \ifin@#2\relax\fi}}
2405 \def\bbl@scswitch{%
2406 \bbl@forlang\bbl@tempa{%
2407 \ifx\bbl@G\@empty\else
2408 \ifx\SetString\@gobbletwo\else
2409 \edef\bbl@GL{\bbl@G\bbl@tempa}%
2410 \bbl@xin@{\bbl@GL,}{,\bbl@screset,}%
2411 \ifin@\else
2412 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
2413 \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
2414 \fi
2415 \fi
2416 \fi}}
2417 \AtEndOfPackage{%
2418 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\#2}}}%
2419 \let\bbl@scswitch\relax}
2420 \@onlypreamble\EndBabelCommands
2421 \def\EndBabelCommands{%
2422 \bbl@usehooks{stopcommands}{}%
2423 \endgroup
2424 \endgroup
2425 \bbl@scafter}
2426 \let\bbl@endcommands\EndBabelCommands

```

Now we define commands to be used inside `\StartBabelCommands`.

**Strings** The following macro is the actual definition of `\SetString` when it is “active” First save the “switcher”. Create it if undefined. Strings are defined only if undefined (ie, like `\providescommand`). With the event `stringprocess` you can preprocess the string by manipulating the value of `\BabelString`. If there are several hooks assigned to this event, preprocessing is done in the same order as defined. Finally, the string is set.

```

2427 \def\bbl@setstring#1#2{%
2428 \bbl@forlang\bbl@tempa{%
2429 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2430 \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2431 {\global\expandafter % TODO - con \bbl@exp ?
2432 \bbl@add\csname\bbl@G\bbl@tempa\expandafter\endcsname\expandafter
2433 {\expandafter\bbl@scset\expandafter#1\csname\bbl@LC\endcsname}}}%

```

```

2434 {}%
2435 \def\BabelString{#2}%
2436 \bbl@usehooks{stringprocess}{}%
2437 \expandafter\bbl@stringdef
2438 \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`.

```

2439 \ifx\bbl@opt@strings\relax
2440 \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2441 \bbl@patchuclc
2442 \let\bbl@encoded\relax
2443 \def\bbl@encoded@uclc#1{%
2444   \@inmathwarn#1%
2445   \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2446     \expandafter\ifx\csname ?\string#1\endcsname\relax
2447       \TextSymbolUnavailable#1%
2448     \else
2449       \csname ?\string#1\endcsname
2450     \fi
2451   \else
2452     \csname\cf@encoding\string#1\endcsname
2453   \fi}
2454 \else
2455   \def\bbl@scset#1#2{\def#1{#2}}
2456 \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.

```

2457 << *Macros local to BabelCommands >> ≡
2458 \def\SetStringLoop##1##2{%
2459   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2460   \count@\z@
2461   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2462     \advance\count@\@ne
2463     \toks@\expandafter{\bbl@tempa}%
2464     \bbl@exp{%
2465       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2466       \count@=\the\count@\relax}}}%
2467 <</Macros local to BabelCommands >>

```

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

```

2468 \def\bbl@aftercmds#1{%
2469   \toks@\expandafter{\bbl@scafter#1}%
2470   \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.

```

2471 << *Macros local to BabelCommands >> ≡
2472 \newcommand\SetCase[3][{}%
2473   \bbl@patchuclc

```

```

2474 \bbl@forlang\bbl@tempa{%
2475 \expandafter\bbl@encstring
2476 \csname\bbl@tempa @bbl@ucl\endcsname{\bbl@tempa##1}%
2477 \expandafter\bbl@encstring
2478 \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2479 \expandafter\bbl@encstring
2480 \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2481 <</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.

```

2482 <<*Macros local to BabelCommands>> ≡
2483 \newcommand\SetHyphenMap[1]{%
2484 \bbl@forlang\bbl@tempa{%
2485 \expandafter\bbl@stringdef
2486 \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
2487 <</Macros local to BabelCommands>>

```

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

```

2488 \newcommand\BabelLower[2]{% one to one.
2489 \ifnum\lccode#1=#2\else
2490 \babel@savevariable{\lccode#1}%
2491 \lccode#1=#2\relax
2492 \fi}
2493 \newcommand\BabelLowerMM[4]{% many-to-many
2494 \@tempcnta=#1\relax
2495 \@tempcntb=#4\relax
2496 \def\bbl@tempa{%
2497 \ifnum\@tempcnta>#2\else
2498 \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2499 \advance\@tempcnta#3\relax
2500 \advance\@tempcntb#3\relax
2501 \expandafter\bbl@tempa
2502 \fi}%
2503 \bbl@tempa}
2504 \newcommand\BabelLowerMO[4]{% many-to-one
2505 \@tempcnta=#1\relax
2506 \def\bbl@tempa{%
2507 \ifnum\@tempcnta>#2\else
2508 \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
2509 \advance\@tempcnta#3
2510 \expandafter\bbl@tempa
2511 \fi}%
2512 \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

2513 <<*More package options>> ≡
2514 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
2515 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
2516 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
2517 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
2518 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
2519 <</More package options>>

```

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

```

2520 \AtEndOfPackage{%
2521 \ifx\bbl@opt@hyphenmap\@undefined
2522 \bbl@xin@{,}{\bbl@language@opts}%

```

```

2523 \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
2524 \fi}

```

## 9.11 Macros common to a number of languages

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

```

2525 \bbl@trace{Macros related to glyphs}
2526 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2527 \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2528 \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.

```

2529 \def\save@sf@q#1{\leavevmode
2530 \begingroup
2531 \edef\SF{\spacefactor\the\spacefactor}#1\SF
2532 \endgroup}

```

## 9.12 Making glyphs available

This section makes a number of glyphs available that either do not exist in the OT1 encoding and have to be ‘faked’, or that are not accessible through `T1enc.def`.

### 9.12.1 Quotation marks

`\quotedblbase` In the T1 encoding the opening double quote at the baseline is available as a separate character, accessible via `\quotedblbase`. In the OT1 encoding it is not available, therefore we make it available by lowering the normal open quote character to the baseline.

```

2533 \ProvideTextCommand{\quotedblbase}{OT1}{%
2534 \save@sf@q{\set@low@box{\textquotedblright\}%
2535 \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.

```

2536 \ProvideTextCommandDefault{\quotedblbase}{%
2537 \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2538 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2539 \save@sf@q{\set@low@box{\textquoteright\}%
2540 \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.

```

2541 \ProvideTextCommandDefault{\quotesinglbase}{%
2542 \UseTextSymbol{OT1}{\quotesinglbase}}

```

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

```

2543 \ProvideTextCommand{\guillemetleft}{OT1}{%
2544 \ifmmode
2545 \ll
2546 \else
2547 \save@sf@q{\nobreak
2548 \raise.2ex\hbox{\scriptscriptstyle\ll}\bbl@allowhyphens}%

```

```

2549 \fi}
2550 \ProvideTextCommand{\guillemetright}{OT1}{%
2551 \ifmmode
2552 \gg
2553 \else
2554 \save@sf@q{\nobreak
2555 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2556 \fi}
2557 \ProvideTextCommand{\guillemotleft}{OT1}{%
2558 \ifmmode
2559 \ll
2560 \else
2561 \save@sf@q{\nobreak
2562 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2563 \fi}
2564 \ProvideTextCommand{\guillemotright}{OT1}{%
2565 \ifmmode
2566 \gg
2567 \else
2568 \save@sf@q{\nobreak
2569 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2570 \fi}

```

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

```

2571 \ProvideTextCommandDefault{\guillemetleft}{%
2572 \UseTextSymbol{OT1}{\guillemetleft}}
2573 \ProvideTextCommandDefault{\guillemetright}{%
2574 \UseTextSymbol{OT1}{\guillemetright}}
2575 \ProvideTextCommandDefault{\guillemotleft}{%
2576 \UseTextSymbol{OT1}{\guillemotleft}}
2577 \ProvideTextCommandDefault{\guillemotright}{%
2578 \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

\guilsinglright 2579 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2580 \ifmmode
2581 <%
2582 \else
2583 \save@sf@q{\nobreak
2584 \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2585 \fi}
2586 \ProvideTextCommand{\guilsinglright}{OT1}{%
2587 \ifmmode
2588 >%
2589 \else
2590 \save@sf@q{\nobreak
2591 \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2592 \fi}

```

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

```

2593 \ProvideTextCommandDefault{\guilsinglleft}{%
2594 \UseTextSymbol{OT1}{\guilsinglleft}}
2595 \ProvideTextCommandDefault{\guilsinglright}{%
2596 \UseTextSymbol{OT1}{\guilsinglright}}

```

### 9.12.2 Letters

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

```
2597 \DeclareTextCommand{\ij}{OT1}{%
2598   i\kern-0.02em\bbl@allowhyphens j}
2599 \DeclareTextCommand{\IJ}{OT1}{%
2600   I\kern-0.02em\bbl@allowhyphens J}
2601 \DeclareTextCommand{\ij}{T1}{\char188}
2602 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2603 \ProvideTextCommandDefault{\ij}{%
2604   \UseTextSymbol{OT1}{\ij}}
2605 \ProvideTextCommandDefault{\IJ}{%
2606   \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 Stipčević Mario, (stipcevic@olimp.irb.hr).

```
2607 \def\crrtic@{\hrule height0.1ex width0.3em}
2608 \def\crrtic@{\hrule height0.1ex width0.33em}
2609 \def\ddj@{%
2610   \setbox0\hbox{d}\dimen@=\ht0
2611   \advance\dimen@1ex
2612   \dimen@.45\dimen@
2613   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2614   \advance\dimen@ii.5ex
2615   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2616 \def\DDJ@{%
2617   \setbox0\hbox{D}\dimen@=.55\ht0
2618   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2619   \advance\dimen@ii.15ex % correction for the dash position
2620   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2621   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2622   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2623 %
2624 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2625 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2626 \ProvideTextCommandDefault{\dj}{%
2627   \UseTextSymbol{OT1}{\dj}}
2628 \ProvideTextCommandDefault{\DJ}{%
2629   \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.

```
2630 \DeclareTextCommand{\SS}{OT1}{\SS}
2631 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

### 9.12.3 Shorthands for quotation marks

Shorthands are provided for a number of different quotation marks, which make them usable both outside and inside mathmode. They are defined with

`\ProvideTextCommandDefault`, but this is very likely not required because their definitions are based on encoding-dependent macros.

`\glq` The ‘german’ single quotes.

```
\grq 2632 \ProvideTextCommandDefault{\glq}{%
2633 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2634 \ProvideTextCommand{\grq}{T1}{%
2635 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}%
2636 \ProvideTextCommand{\grq}{TU}{%
2637 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2638 \ProvideTextCommand{\grq}{OT1}{%
2639 \save@sf@q{\kern-.0125em
2640 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2641 \kern.07em\relax}}
2642 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

`\glqq` The ‘german’ double quotes.

```
\grqq 2643 \ProvideTextCommandDefault{\glqq}{%
2644 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2645 \ProvideTextCommand{\grqq}{T1}{%
2646 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2647 \ProvideTextCommand{\grqq}{TU}{%
2648 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2649 \ProvideTextCommand{\grqq}{OT1}{%
2650 \save@sf@q{\kern-.07em
2651 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2652 \kern.07em\relax}}
2653 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq` The ‘french’ single guillemets.

```
\frq 2654 \ProvideTextCommandDefault{\flq}{%
2655 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}%
2656 \ProvideTextCommandDefault{\frq}{%
2657 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq 2658 \ProvideTextCommandDefault{\flqq}{%
2659 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}%
2660 \ProvideTextCommandDefault{\frqq}{%
2661 \textormath{\guillemetright}{\mbox{\guillemetright}}}
```

#### 9.12.4 Umlauts and tremas

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

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

```
2662 \def\umlauthigh{%
```

```

2663 \def\bbl@umlauta##1{\leavevmode\bgroup%
2664 \expandafter\accent\csname\fontencoding dqpos\endcsname
2665 ##1\bbl@allowhyphens\egroup}%
2666 \let\bbl@umlaute\bbl@umlauta}
2667 \def\umlautlow{%
2668 \def\bbl@umlauta{\protect\lower@umlaut}}
2669 \def\umlautelowlow{%
2670 \def\bbl@umlaute{\protect\lower@umlaut}}
2671 \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.

```

2672 \expandafter\ifx\csname U@D\endcsname\relax
2673 \csname newdimen\endcsname\U@D
2674 \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.

```

2675 \def\lower@umlaut#1{%
2676 \leavevmode\bgroup
2677 \U@D 1ex%
2678 {\setbox\z@\hbox{%
2679 \expandafter\char\csname\fontencoding dqpos\endcsname}%
2680 \dimen@ -.45ex\advance\dimen@\ht\z@
2681 \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2682 \expandafter\accent\csname\fontencoding dqpos\endcsname
2683 \fontdimen5\font\U@D #1%
2684 \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).

```

2685 \AtBeginDocument{%
2686 \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2687 \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2688 \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2689 \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{i}}%
2690 \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
2691 \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
2692 \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
2693 \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2694 \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2695 \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
2696 \DeclareTextCompositeCommand{\}{OT1}{U}{\bbl@umlauta{U}}%

```



Finally, make sure the default hyphenrules are defined (even if empty).

```
2697 \ifx\l@english\@undefined
2698   \chardef\l@english\z@
2699 \fi
```

## 9.13 Layout

### Work in progress.

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

```
2700 \bbl@trace{Bidi layout}
2701 \providecommand\IfBabelLayout[3]{#3}%
2702 \newcommand\BabelPatchSection[1]{%
2703   \@ifundefined{#1}{}{%
2704     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2705     \@namedef{#1}{%
2706       \ifstar{\bbl@presec@s{#1}}%
2707         {\@dblarg{\bbl@presec@x{#1}}}}}%
2708 \def\bbl@presec@x#1[#2]#3{%
2709   \bbl@exp{%
2710     \\\select@language@x{\bbl@main@language}%
2711     \\\bbl@cs{sspre@#1}%
2712     \\\bbl@cs{ss@#1}%
2713     [\\foreignlanguage{\languagename}{\unexpanded{#2}}]%
2714     {\foreignlanguage{\languagename}{\unexpanded{#3}}}%
2715     \\\select@language@x{\languagename}}}%
2716 \def\bbl@presec@s#1#2{%
2717   \bbl@exp{%
2718     \\\select@language@x{\bbl@main@language}%
2719     \\\bbl@cs{sspre@#1}%
2720     \\\bbl@cs{ss@#1}*%
2721     {\foreignlanguage{\languagename}{\unexpanded{#2}}}%
2722     \\\select@language@x{\languagename}}}%
2723 \IfBabelLayout{sectioning}%
2724   {\BabelPatchSection{part}%
2725    \BabelPatchSection{chapter}%
2726    \BabelPatchSection{section}%
2727    \BabelPatchSection{subsection}%
2728    \BabelPatchSection{subsubsection}%
2729    \BabelPatchSection{paragraph}%
2730    \BabelPatchSection{subparagraph}}%
2731   \def\babel@toc#1{%
2732     \select@language@x{\bbl@main@language}}}%
2733 \IfBabelLayout{captions}%
2734   {\BabelPatchSection{caption}}}%
```

## 9.14 Load engine specific macros

```
2735 \bbl@trace{Input engine specific macros}
2736 \ifcase\bbl@engine
2737   \input txtbabel.def
2738 \or
2739   \input luababel.def
2740 \or
2741   \input xebabel.def
2742 \fi
```

## 9.15 Creating and modifying languages

`\babelprovide` is a general purpose tool for creating and modifying languages. It creates the language infrastructure, and loads, if requested, an ini file. It may be used in conjunction to previously loaded ldf files.

```
2743 \bbl@trace{Creating languages and reading ini files}
2744 \newcommand\babelprovide[2][{}]{%
2745   \let\bbl@savelangname\language
2746   \edef\bbl@savelocaleid{\the\localeid}%
2747   % Set name and locale id
2748   \edef\language{#2}%
2749   % \global\@namedef{\bbl@lcname@#2}{#2}%
2750   \bbl@id@assign
2751   \let\bbl@KVP@captions\@nil
2752   \let\bbl@KVP@import\@nil
2753   \let\bbl@KVP@main\@nil
2754   \let\bbl@KVP@script\@nil
2755   \let\bbl@KVP@language\@nil
2756   \let\bbl@KVP@hyphenrules\@nil % only for provide@new
2757   \let\bbl@KVP@mapfont\@nil
2758   \let\bbl@KVP@maparabic\@nil
2759   \let\bbl@KVP@mapdigits\@nil
2760   \let\bbl@KVP@intraspace\@nil
2761   \let\bbl@KVP@intrapenalty\@nil
2762   \let\bbl@KVP@onchar\@nil
2763   \let\bbl@KVP@alph\@nil
2764   \let\bbl@KVP@Alph\@nil
2765   \let\bbl@KVP@info\@nil % Ignored with import? Or error/warning?
2766   \bbl@forkv{#1}{% TODO - error handling
2767     \in@{/}{##1}%
2768     \ifin@
2769       \bbl@renewinikey##1\@{##2}%
2770     \else
2771       \bbl@csarg\def{KVP@##1}{##2}%
2772     \fi}%
2773   % == import, captions ==
2774   \ifx\bbl@KVP@import\@nil\else
2775     \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
2776     {\ifx\bbl@initoload\relax
2777       \begingroup
2778         \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2779         \InputIfFileExists{babel-#2.tex}{}}%
2780       \endgroup
2781     \else
2782       \xdef\bbl@KVP@import{\bbl@initoload}%
2783     \fi}%
2784   {}%
2785 \fi
2786 \ifx\bbl@KVP@captions\@nil
2787   \let\bbl@KVP@captions\bbl@KVP@import
2788 \fi
2789 % Load ini
2790 \bbl@ifunset{date#2}%
2791   {\bbl@provide@new{#2}}%
2792   {\bbl@ifblank{#1}%
2793     {\bbl@error
2794       {If you want to modify `#2' you must tell how in\\%
2795       the optional argument. See the manual for the\\%
```

```

2796         available options.}%
2797     {Use this macro as documented}}%
2798     {\bbl@provide@renew{#2}}}%
2799 % Post tasks
2800 \bbl@exp{\babelensure[exclude=\\today]{#2}}%
2801 \bbl@ifunset{\bbl@ensure@language}%
2802     {\bbl@exp{%
2803         \\DeclareRobustCommand\<bbl@ensure@language>[1]{%
2804             \\foreignlanguage{language}%
2805             {###1}}}%
2806     }%
2807 \bbl@exp{%
2808     \\bbl@toGLOBAL\<bbl@ensure@language>%
2809     \\bbl@toGLOBAL\<bbl@ensure@language\space>%
2810 % At this point all parameters are defined if 'import'. Now we
2811 % execute some code depending on them. But what about if nothing was
2812 % imported? We just load the very basic parameters: ids and a few
2813 % more.
2814 \bbl@ifunset{\bbl@lname{#2}}% TODO. Duplicated
2815     {\def\BabelBeforeIni##1##2{%
2816         \begingroup
2817         \catcode`\[=12 \catcode`\]=12 \catcode`\==12
2818         \catcode`\;=12 \catcode`\|=12 %
2819         \let\bbl@ini@captions@aux\@gobbletwo
2820         \def\bbl@inidate ###1.###2.###3.###4\relax ###5####6{%
2821             \bbl@read@ini{##1}{basic data}%
2822             \bbl@exportkey{chrng}{characters.ranges}{}%
2823             \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2824             \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2825             \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2826             \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
2827             \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2828             \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2829             \bbl@exportkey{hyoth}{typography.hyphenate.other}{}%
2830             \bbl@exportkey{intsp}{typography.intraspaces}{}%
2831             \ifx\bbl@initoload\relax\endinput\fi
2832         \endgroup}%
2833         \begingroup % boxed, to avoid extra spaces:
2834         \ifx\bbl@initoload\relax
2835             \setbox\z@\hbox{\InputIfFileExists{babel-#2.tex}{}}%
2836         \else
2837             \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}%
2838         \fi
2839         \endgroup}%
2840     }%
2841 % == script, language ==
2842 % Override the values from ini or defines them
2843 \ifx\bbl@KVP@script\@nil\else
2844     \bbl@csarg\edef\sname{#2}{\bbl@KVP@script}%
2845 \fi
2846 \ifx\bbl@KVP@language\@nil\else
2847     \bbl@csarg\edef\lname{#2}{\bbl@KVP@language}%
2848 \fi
2849 % == onchar ==
2850 \ifx\bbl@KVP@onchar\@nil\else
2851     \bbl@luahyphenate
2852     \directlua{
2853         if Babel.locale_mapped == nil then
2854             Babel.locale_mapped = true

```

```

2855     Babel.linebreaking.add_before(Babel.locale_map)
2856     Babel.loc_to_scr = {}
2857     Babel.chr_to_loc = Babel.chr_to_loc or {}
2858   end}%
2859   \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2860   \ifin@
2861     \ifx\bbl@starthyphens\undefined % Needed if no explicit selection
2862       \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
2863     \fi
2864     \bbl@exp{\bbl@add\bbl@starthyphens
2865       {\bbl@patterns@lua{\language}}}%
2866     % TODO - error/warning if no script
2867     \directlua{
2868       if Babel.script_blocks['\bbl@cl{sbc}'] then
2869         Babel.loc_to_scr[\the\localeid] =
2870           Babel.script_blocks['\bbl@cl{sbc}']
2871         Babel.locale_props[\the\localeid].lc = \the\localeid\space
2872         Babel.locale_props[\the\localeid].lg = \the\nameuse{1@\language}\space
2873       end
2874     }%
2875   \fi
2876   \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2877   \ifin@
2878     \bbl@ifunset{\bbl@lsys@\language}{\bbl@provide@lsys{\language}}}%
2879     \bbl@ifunset{\bbl@wdir@\language}{\bbl@provide@dirs{\language}}}%
2880     \directlua{
2881       if Babel.script_blocks['\bbl@cl{sbc}'] then
2882         Babel.loc_to_scr[\the\localeid] =
2883           Babel.script_blocks['\bbl@cl{sbc}']
2884       end}%
2885     \ifx\bbl@mapselect\undefined
2886       \AtBeginDocument{%
2887         \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
2888         {\selectfont}}%
2889       \def\bbl@mapselect{%
2890         \let\bbl@mapselect\relax
2891         \edef\bbl@prefontid{\fontid\font}}%
2892       \def\bbl@mapdir##1{%
2893         {\def\language{##1}%
2894           \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2895           \bbl@switchfont
2896           \directlua{
2897             Babel.locale_props[\the\csname bbl@id@##1\endcsname]
2898               [\bbl@prefontid] = \fontid\font\space}}}%
2899       \fi
2900       \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
2901     \fi
2902     % TODO - catch non-valid values
2903   \fi
2904   % == mapfont ==
2905   % For bidi texts, to switch the font based on direction
2906   \ifx\bbl@KVP@mapfont\@nil\else
2907     \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}}%
2908     {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\
2909       mapfont. Use 'direction'.%
2910       {See the manual for details.}}}%
2911     \bbl@ifunset{\bbl@lsys@\language}{\bbl@provide@lsys{\language}}}%
2912     \bbl@ifunset{\bbl@wdir@\language}{\bbl@provide@dirs{\language}}}%
2913     \ifx\bbl@mapselect\undefined

```

```

2914 \AtBeginDocument{%
2915   \expandafter\bbbl@add\csname selectfont \endcsname{\bbbl@mapselect}}%
2916   {\selectfont}}%
2917 \def\bbbl@mapselect{%
2918   \let\bbbl@mapselect\relax
2919   \edef\bbbl@prefontid{\fontid\font}}%
2920 \def\bbbl@mapdir##1{%
2921   {\def\language{##1}%
2922     \let\bbbl@ifrestoring\@firstoftwo % avoid font warning
2923     \bbbl@switchfont
2924     \directlua{Babel.fontmap
2925       [\the\csname bbl@wdir@##1\endcsname]%
2926       [\bbbl@prefontid]=\fontid\font}}}%
2927 \fi
2928 \bbbl@exp{\bbbl@add\bbbl@mapselect{\bbbl@mapdir{\language}}}%
2929 \fi
2930 % == intraspace, intrapenalty ==
2931 % For CJK, East Asian, Southeast Asian, if interspace in ini
2932 \ifx\bbbl@KVP@intraspace\@nil\else % We can override the ini or set
2933   \bbbl@csarg\edef{intsp@#2}{\bbbl@KVP@intraspace}%
2934 \fi
2935 \bbbl@provide@intraspace
2936 % == hyphenate.other ==
2937 \bbbl@ifunset{\bbbl@hyoth@\language}{}%
2938   {\bbbl@csarg\bbbl@replace{\hyoth@\language}{ }{ },}%
2939   \bbbl@startcommands*{\language}{}%
2940   \bbbl@csarg\bbbl@foreach{\hyoth@\language}{%
2941     \ifcase\bbbl@engine
2942       \ifnum##1<257
2943         \SetHyphenMap{\BabelLower{##1}{##1}}%
2944       \fi
2945       \else
2946         \SetHyphenMap{\BabelLower{##1}{##1}}%
2947       \fi}%
2948   \bbbl@endcommands}%
2949 % == maparabic ==
2950 % Native digits, if provided in ini (TeX level, xe and lua)
2951 \ifcase\bbbl@engine\else
2952   \bbbl@ifunset{\bbbl@dgnat@\language}{}%
2953     {\expandafter\ifx\csname bbl@dgnat@\language\endcsname\@empty\else
2954       \expandafter\expandafter\expandafter
2955       \bbbl@setdigits\csname bbl@dgnat@\language\endcsname
2956       \ifx\bbbl@KVP@maparabic\@nil\else
2957         \ifx\bbbl@latinarabic\@undefined
2958           \expandafter\let\expandafter\@arabic
2959           \csname bbl@counter@\language\endcsname
2960         \else % ie, if layout=counters, which redefines \@arabic
2961           \expandafter\let\expandafter\bbbl@latinarabic
2962           \csname bbl@counter@\language\endcsname
2963         \fi
2964       \fi
2965     \fi}%
2966 \fi
2967 % == mapdigits ==
2968 % Native digits (lua level).
2969 \ifodd\bbbl@engine
2970   \ifx\bbbl@KVP@mapdigits\@nil\else
2971     \bbbl@ifunset{\bbbl@dgnat@\language}{}%
2972     {\RequirePackage{luatexbase}%

```

```

2973     \bbl@activate@preotf
2974     \directlua{
2975         Babel = Babel or {}  %% -> presets in luababel
2976         Babel.digits_mapped = true
2977         Babel.digits = Babel.digits or {}
2978         Babel.digits[\the\localeid] =
2979             table.pack(string.utfvalue('\bbl@cl{dgnat}'))
2980         if not Babel.numbers then
2981             function Babel.numbers(head)
2982                 local LOCALE = luatexbase.registernumber'bbl@attr@locale'
2983                 local GLYPH = node.id'glyph'
2984                 local inmath = false
2985                 for item in node.traverse(head) do
2986                     if not inmath and item.id == GLYPH then
2987                         local temp = node.get_attribute(item, LOCALE)
2988                         if Babel.digits[temp] then
2989                             local chr = item.char
2990                             if chr > 47 and chr < 58 then
2991                                 item.char = Babel.digits[temp][chr-47]
2992                             end
2993                         end
2994                         elseif item.id == node.id'math' then
2995                             inmath = (item.subtype == 0)
2996                         end
2997                     end
2998                 return head
2999             end
3000         end
3001     } }%
3002     \fi
3003     \fi
3004     % == alph, Alph ==
3005     % What if extras<lang> contains a \babel@save\@alph? It won't be
3006     % restored correctly when exiting the language, so we ignore
3007     % this change with the \bbl@alph@saved trick.
3008     \ifx\bbl@KVP@alph\@nil\else
3009         \toks@\expandafter\expandafter\expandafter{%
3010             \csname extras\language\endcsname}%
3011         \bbl@exp{%
3012             \def\<extras\language>{%
3013                 \let\\bbl@alph@saved\\@alph
3014                 \the\toks@
3015                 \let\\@alph\\bbl@alph@saved
3016                 \\babel@save\\@alph
3017                 \let\\@alph\<bbl@cntr@\bbl@KVP@alph @\language>}}%
3018         \fi
3019     \ifx\bbl@KVP@Alph\@nil\else
3020         \toks@\expandafter\expandafter\expandafter{%
3021             \csname extras\language\endcsname}%
3022         \bbl@exp{%
3023             \def\<extras\language>{%
3024                 \let\\bbl@Alph@saved\\@Alph
3025                 \the\toks@
3026                 \let\\@Alph\\bbl@Alph@saved
3027                 \\babel@save\\@Alph
3028                 \let\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\language>}}%
3029         \fi
3030     % == require.babel in ini ==
3031     % To load or reload the babel-*.tex, if require.babel in ini

```



```

3085         \bbl@exp{%
3086             \\\SetString\\##1{%
3087                 \\\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
3088         \expandafter\bbl@tempb
3089         \fi}%
3090     \expandafter\bbl@tempb\bbl@captionslist\@empty
3091 \else
3092     \ifx\bbl@initoload\relax
3093         \bbl@read@ini{\bbl@KVP@captions}{data}% Here letters cat = 11
3094     \else
3095         \bbl@read@ini{\bbl@initoload}{data}% Here all letters cat = 11
3096     \fi
3097     \bbl@after@ini
3098     \bbl@savestrings
3099 \fi
3100 \StartBabelCommands*{#1}{date}%
3101 \ifx\bbl@KVP@import\@nil
3102     \bbl@exp{%
3103         \\\SetString\\today{\bbl@nocaption{today}{#1today}}}%
3104 \else
3105     \bbl@savetoday
3106     \bbl@savedate
3107 \fi
3108 \bbl@endcommands
3109 \bbl@ifunset{bbl@lname@#1}%      TODO. Duplicated
3110 {\def\BabelBeforeIni##1##2{%
3111     \begingroup
3112         \catcode`\[=12 \catcode`\]=12 \catcode`\==12
3113         \catcode`\;=12 \catcode`\|=12 %
3114         \let\bbl@ini@captions@aux\@gobbletwo
3115         \def\bbl@inidate ###1.###2.###3.###4\relax ###5###6{%
3116             \bbl@read@ini{##1}{basic data}%
3117             \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3118             \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3119             \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3120             \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3121             \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3122             \bbl@exportkey{hyoth}{typography.hyphenate.other}{}%
3123             \bbl@exportkey{intsp}{typography.intraspace}{}%
3124             \bbl@exportkey{chrng}{characters.ranges}{}%
3125             \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3126             \ifx\bbl@initoload\relax\endinput\fi
3127         \endgroup}%
3128     \begingroup      % boxed, to avoid extra spaces:
3129     \ifx\bbl@initoload\relax
3130         \setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}}}%
3131     \else
3132         \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}}%
3133     \fi
3134     \endgroup}%
3135     {}}%
3136 \bbl@exp{%
3137     \gdef\<#1hyphenmins>{%
3138         {\bbl@ifunset{bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
3139         {\bbl@ifunset{bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
3140 \bbl@provide@hyphens{#1}%
3141 \ifx\bbl@KVP@main\@nil\else
3142     \expandafter\main@language\expandafter{#1}%
3143 \fi}

```



```

3144 \def\bbl@provide@renew#1{%
3145   \ifx\bbl@KVP@captions\@nil\else
3146     \StartBabelCommands*{#1}{captions}%
3147     \bbl@read@ini{\bbl@KVP@captions}{data}%   Here all letters cat = 11
3148     \bbl@after@ini
3149     \bbl@savestrings
3150     \EndBabelCommands
3151   \fi
3152   \ifx\bbl@KVP@import\@nil\else
3153     \StartBabelCommands*{#1}{date}%
3154     \bbl@savetoday
3155     \bbl@savedate
3156     \EndBabelCommands
3157   \fi
3158   % == hyphenrules ==
3159   \bbl@provide@hyphens{#1}}

```

The hyphenrules option is handled with an auxiliary macro.

```

3160 \def\bbl@provide@hyphens#1{%
3161   \let\bbl@tempa\relax
3162   \ifx\bbl@KVP@hyphenrules\@nil\else
3163     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
3164     \bbl@foreach\bbl@KVP@hyphenrules{%
3165       \ifx\bbl@tempa\relax    % if not yet found
3166         \bbl@ifsamestring{##1}{+}%
3167         {{\bbl@exp{\addlanguage\<l@##1>}}}%
3168         {}}%
3169         \bbl@ifunset{l@##1}%
3170         {}}%
3171         {\bbl@exp{\let\bbl@tempa\<l@##1>}}}%
3172     \fi}%
3173   \fi
3174   \ifx\bbl@tempa\relax %      if no opt or no language in opt found
3175     \ifx\bbl@KVP@import\@nil
3176       \ifx\bbl@initoload\relax\else
3177         \bbl@exp{%              and hyphenrules is not empty
3178           \\bbl@ifblank{\bbl@cs{hyphr@#1}}%
3179           {}}%
3180           {\let\\bbl@tempa\<l@bbl@cl{hyphr}>}}}%
3181       \fi
3182     \else % if importing
3183       \bbl@exp{%              and hyphenrules is not empty
3184         \\bbl@ifblank{\bbl@cs{hyphr@#1}}%
3185         {}}%
3186         {\let\\bbl@tempa\<l@bbl@cl{hyphr}>}}}%
3187     \fi
3188   \fi
3189   \bbl@ifunset\bbl@tempa{%      ie, relax or undefined
3190     {\bbl@ifunset{l@#1}%        no hyphenrules found - fallback
3191       {\bbl@exp{\adddialect\<l@#1>\language}}%
3192       {}}%
3193       {\bbl@exp{\adddialect\<l@#1>\bbl@tempa}}}% found in opt list or ini
3194   }

```

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.

```

3195 \ifx\bbl@readstream\@undefined
3196   \csname newread\endcsname\bbl@readstream
3197 \fi

```

```

3198 \def\bbbl@inipreread#1=#2\@@{%
3199   \bbbl@trim@def\bbbl@tempa{#1}% Redundant below !!
3200   \bbbl@trim\toks@{#2}%
3201   % Move trims here ??
3202   \bbbl@ifunset{\bbbl@KVP@\bbbl@section/\bbbl@tempa}%
3203   {\bbbl@exp{%
3204     \\g@addto@macro\\bbbl@inidata{%
3205       \\bbbl@elt{\bbbl@section}{\bbbl@tempa}{\the\toks@}}}%
3206     \expandafter\bbbl@inireader\bbbl@tempa=#2\@@}%
3207   }%
3208 \def\bbbl@read@ini#1#2{%
3209   \bbbl@csarg\edef\lini@\language\{#1}%
3210   \openin\bbbl@readstream=babel-#1.ini
3211   \ifeof\bbbl@readstream
3212     \bbbl@error
3213     {There is no ini file for the requested language\\%
3214      (#1). Perhaps you misspelled it or your installation\\%
3215      is not complete.}%
3216     {Fix the name or reinstall babel.}%
3217   \else
3218     \bbbl@exp{\def\\bbbl@inidata{\\bbbl@elt{identificacion}{tag.ini}{#1}}}%
3219     \let\bbbl@section\@empty
3220     \let\bbbl@savestrings\@empty
3221     \let\bbbl@savetoday\@empty
3222     \let\bbbl@savestate\@empty
3223     \let\bbbl@inireader\bbbl@iniskip
3224     \bbbl@info{Importing #2 for \language\\%
3225               from babel-#1.ini. Reported}%
3226     \loop
3227     \if \T\ifeof\bbbl@readstream F\fi T\relax % Trick, because inside \loop
3228       \endlinechar\m@ne
3229       \read\bbbl@readstream to \bbbl@line
3230       \endlinechar\^^M
3231       \ifx\bbbl@line\@empty\else
3232         \expandafter\bbbl@iniline\bbbl@line\bbbl@iniline
3233       \fi
3234     \repeat
3235     \bbbl@foreach\bbbl@renewlist{%
3236       \bbbl@ifunset{\bbbl@renew@##1}{\bbbl@inisec[##1]\@@}%
3237     \global\let\bbbl@renewlist\@empty
3238     % Ends last section. See \bbbl@inisec
3239     \def\bbbl@elt##1##2{\bbbl@inireader##1=#2\@@}%
3240     \bbbl@cs{renew@\bbbl@section}%
3241     \global\bbbl@csarg\let{renew@\bbbl@section}\relax
3242     \bbbl@cs{secpost@\bbbl@section}%
3243     \bbbl@csarg{\global\expandafter\let}{inidata@\language}\bbbl@inidata
3244     \bbbl@exp{\\bbbl@add@list\\bbbl@ini@loaded{\language}}%
3245     \bbbl@tglobal\bbbl@ini@loaded
3246   \fi}
3247 \def\bbbl@iniline#1\bbbl@iniline{%
3248   \@ifnextchar[\bbbl@inisec{\@ifnextchar;\bbbl@iniskip\bbbl@inipreread}#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. The secpost “hook” is used only by ‘identification’, while secpre only by date.gregorian.licr.

```

3249 \def\bbbl@iniskip#1\@@{%      if starts with ;
3250 \def\bbbl@inisec[#1]#2\@@{%   if starts with opening bracket

```

```

3251 \def\bbl@elt##1##2{%
3252   \expandafter\toks@\expandafter{%
3253     \expandafter{\bbl@section}{##1}{##2}}%
3254   \bbl@exp{%
3255     \g@addto@macro\\bbl@inidata{\bbl@elt\the\toks@}}%
3256   \bbl@inireader##1=##2\@@}%
3257 \bbl@cs{renew\bbl@section}%
3258 \global\bbl@csarg\let{renew\bbl@section}\relax
3259 \bbl@cs{secpost\bbl@section}%
3260 % The previous code belongs to the previous section.
3261 % Now start the current one.
3262 \def\bbl@section{#1}%
3263 \def\bbl@elt##1##2{%
3264   \@namedef{bbl@KVP@#1/#1}{}}%
3265 \bbl@cs{renew@#1}%
3266 \bbl@cs{secpre@#1}% pre-section 'hook'
3267 \bbl@ifunset{bbl@inikv@#1}%
3268   {\let\bbl@inireader\bbl@iniskip}%
3269   {\bbl@exp{\let\\bbl@inireader\<bbl@inikv@#1>}}%
3270 \let\bbl@renewlist\@empty
3271 \def\bbl@renewinikv#1/#2\@@#3{%
3272   \bbl@ifunset{bbl@renew@#1}%
3273     {\bbl@add@list\bbl@renewlist{#1}}%
3274     {}}%
3275 \bbl@csarg\bbl@add{renew@#1}{\bbl@elt{#2}{#3}}%

```

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

```

3276 \def\bbl@inikv#1=#2\@@{%      key=value
3277   \bbl@trim@def\bbl@tempa{#1}%
3278   \bbl@trim\toks@{#2}%
3279   \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.

```

3280 \def\bbl@exportkey#1#2#3{%
3281   \bbl@ifunset{bbl@kv@#2}%
3282     {\bbl@csarg\gdef{#1@\language}\{#3}}%
3283     {\expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
3284       \bbl@csarg\gdef{#1@\language}\{#3}}%
3285     \else
3286       \bbl@exp{\global\let\<bbl@#1@\language>\<bbl@kv@#2>}}%
3287     \fi}}

```

Key-value pairs are treated differently depending on the section in the ini file. The following macros are the readers for identification and typography. Note \bbl@secpost@identification is called always (via \bbl@inisec), while \bbl@after@ini must be called explicitly after \bbl@read@ini if necessary.

```

3288 \def\bbl@iniwarning#1{%
3289   \bbl@ifunset{bbl@kv@identification.warning#1}{}}%
3290   {\bbl@warning{%
3291     From babel-\bbl@cs{lini@\language}.ini:\\%
3292     \bbl@cs{kv@identification.warning#1}\\%
3293     Reported }}%
3294 \let\bbl@inikv@identification\bbl@inikv
3295 \def\bbl@secpost@identification{%
3296   \bbl@iniwarning}%
3297   \ifcase\bbl@engine
3298     \bbl@iniwarning{.pdf\latex}%
3299   \or

```

```

3300 \bbl@iniwarning{.lualatex}%
3301 \or
3302 \bbl@iniwarning{.xelatex}%
3303 \fi%
3304 \bbl@exportkey{elname}{identification.name.english}{}%
3305 \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
3306 {\csname bbl@elname@language\endcsname}}%
3307 \bbl@exportkey{lbcpr}{identification.tag.bcp47}{}%
3308 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3309 \bbl@exportkey{esname}{identification.script.name}{}%
3310 \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
3311 {\csname bbl@esname@language\endcsname}}%
3312 \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
3313 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}}
3314 \let\bbl@inikv@typography\bbl@inikv
3315 \let\bbl@inikv@characters\bbl@inikv
3316 \let\bbl@inikv@numbers\bbl@inikv
3317 \def\bbl@inikv@counters#1=#2\@@{%
3318 \def\bbl@tempc{#1}%
3319 \bbl@trim@def{\bbl@tempb*}{#2}%
3320 \in@{.1$}{#1$}%
3321 \ifin@
3322 \bbl@replace\bbl@tempc{.1}{}%
3323 \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}{%
3324 \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3325 \fi
3326 \in@{.F.}{#1}%
3327 \ifin@else\in@{.S.}{#1}\fi
3328 \ifin@
3329 \bbl@csarg\protected@xdef{cntr@#1@\language}{\bbl@tempb*}%
3330 \else
3331 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3332 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3333 \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
3334 \fi}
3335 \def\bbl@after@ini{%
3336 \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3337 \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3338 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3339 \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3340 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3341 \bbl@exportkey{hyoth}{typography.hyphenate.other}{}%
3342 \bbl@exportkey{intsp}{typography.intraspace}{}%
3343 \bbl@exportkey{jstfy}{typography.justify}{w}%
3344 \bbl@exportkey{chrng}{characters.ranges}{}%
3345 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3346 \bbl@exportkey{rqtex}{identification.require.babel}{}%
3347 \bbl@toggle\bbl@savetoday
3348 \bbl@toggle\bbl@savestate}

```

Now captions and captions.licr, depending on the engine. And below also for dates. They rely on a few auxiliary macros. It is expected the ini file provides the complete set in Unicode and LICR, in that order.

```

3349 \ifcase\bbl@engine
3350 \bbl@csarg\def{inikv@captions.licr}#1=#2\@@{%
3351 \bbl@ini@captions@aux{#1}{#2}}
3352 \else
3353 \def\bbl@inikv@captions#1=#2\@@{%
3354 \bbl@ini@captions@aux{#1}{#2}}

```

3355 \fi

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

```
3356 \def\bbl@ini@captions@aux#1#2{%
3357   \bbl@trim@def\bbl@tempa{#1}%
3358   \bbl@ifblank{#2}%
3359   {\bbl@exp{%
3360     \toks@{\bbl@nocaption{\bbl@tempa}{\language\name\bbl@tempa name}}}%
3361   {\bbl@trim\toks@{#2}}}%
3362   \bbl@exp{%
3363     \bbl@add\bbl@savestrings{%
3364       \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 (saved months are just discarded when the LICR section is reached).

TODO. Remove cypypaste pattern.

```
3365 \bbl@csarg\def{inikv@date.gregorian}#1=#2\@@{%      for defaults
3366   \bbl@inidate#1...\relax{#2}}%
3367 \bbl@csarg\def{inikv@date.islamic}#1=#2\@@{%
3368   \bbl@inidate#1...\relax{#2}{islamic}}%
3369 \bbl@csarg\def{inikv@date.hebrew}#1=#2\@@{%
3370   \bbl@inidate#1...\relax{#2}{hebrew}}%
3371 \bbl@csarg\def{inikv@date.persian}#1=#2\@@{%
3372   \bbl@inidate#1...\relax{#2}{persian}}%
3373 \bbl@csarg\def{inikv@date.indian}#1=#2\@@{%
3374   \bbl@inidate#1...\relax{#2}{indian}}%
3375 \ifcase\bbl@engine
3376   \bbl@csarg\def{inikv@date.gregorian.licr}#1=#2\@@{%  override
3377     \bbl@inidate#1...\relax{#2}}%
3378   \bbl@csarg\def{secpre@date.gregorian.licr}{%          discard uni
3379     \ifcase\bbl@engine\let\bbl@savestate@empty\fi}
3380 \fi
3381 % eg: 1=months, 2=wide, 3=1, 4=dummy
3382 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3383   \bbl@trim@def\bbl@tempa{#1.#2}%
3384   \bbl@ifsamestring{\bbl@tempa}{months.wide}%          to savedate
3385   {\bbl@trim@def\bbl@tempa{#3}%
3386     \bbl@trim\toks@{#5}%
3387     \bbl@exp{%
3388       \bbl@add\bbl@savestate{%
3389         \SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}}}%
3390   {\bbl@ifsamestring{\bbl@tempa}{date.long}%            defined now
3391     {\bbl@trim@def\bbl@toreplace{#5}%
3392       \bbl@TG@date
3393       \global\bbl@csarg\let{date@\language}\bbl@toreplace
3394       \bbl@exp{%
3395         \gdef\<\language date>{\protect\<\language date >}%
3396         \gdef\<\language date >####1####2####3{%
3397           \bbl@usedategrouptrue
3398           \<\bbl@ensure@\language>{%
3399             \<\bbl@date@\language>{####1}{####2}{####3}}}%
3400         \bbl@add\bbl@savetoday{%
3401           \SetString\<\today>{%
3402             \<\language date>{\the\year}{\the\month}{\the\day}}}%
3403       }}
```

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.

```

3404 \let\bbl@calendar\@empty
3405 \newcommand\BabelDateSpace{\nobreakspace}
3406 \newcommand\BabelDateDot{.\@}
3407 \newcommand\BabelDated[1]{\number#1}
3408 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3409 \newcommand\BabelDateM[1]{\number#1}
3410 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3411 \newcommand\BabelDateMMM[1]{\%
3412 \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3413 \newcommand\BabelDateY[1]{\number#1}%
3414 \newcommand\BabelDateyy[1]{\%
3415 \ifnum#1<10 0\number#1 %
3416 \else\ifnum#1<100 \number#1 %
3417 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3418 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3419 \else
3420 \bbl@error
3421 {Currently two-digit years are restricted to the\
3422 range 0-9999.}%
3423 {There is little you can do. Sorry.}%
3424 \fi\fi\fi}
3425 \newcommand\BabelDateyyyy[1]{\number#1} % FIXME - add leading 0
3426 \def\bbl@replace@finish@iii#1{%
3427 \bbl@exp{\def\#1####1####2####3\the\toks@}}
3428 \def\bbl@TG@date{%
3429 \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3430 \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3431 \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3432 \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3433 \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3434 \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3435 \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMM{####2}}%
3436 \bbl@replace\bbl@toreplace{[y]}{\BabelDateY{####1}}%
3437 \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3438 \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3439 \bbl@replace\bbl@toreplace{[y|]}{\bbl@datecctr[####1|]}%
3440 \bbl@replace\bbl@toreplace{[m|]}{\bbl@datecctr[####2|]}%
3441 \bbl@replace\bbl@toreplace{[d|]}{\bbl@datecctr[####3|]}%
3442 % Note after \bbl@replace \toks@ contains the resulting string.
3443 % TODO - Using this implicit behavior doesn't seem a good idea.
3444 \bbl@replace@finish@iii\bbl@toreplace}
3445 \def\bbl@datecctr[#1|#2]{\localenumeral{#2}{#1}}

```

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

```

3446 \def\bbl@provide@lsys#1{%
3447 \bbl@ifunset{\bbl@lname@#1}%
3448 {\bbl@ini@basic{#1}}%
3449 {}%
3450 \bbl@csarg\let{lsys@#1}\@empty
3451 \bbl@ifunset{\bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3452 \bbl@ifunset{\bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3453 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3454 \bbl@ifunset{\bbl@lname@#1}{%
3455 {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3456 \ifcase\bbl@engine\or\or
3457 \bbl@ifunset{\bbl@prehc@#1}{}%

```

```

3458      {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3459      }%
3460      {\bbl@csarg\bbl@add@list{lsys@#1}{HyphenChar="200B}}}%
3461 \fi
3462 \bbl@csarg\bbl@toglobal{lsys@#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 (which means any code in it must be skipped, too).

```

3463 \def\bbl@ini@basic#1{%
3464   \def\BabelBeforeIni##1##2{%
3465     \begingroup
3466     \bbl@add\bbl@secpost@identification{\closein\bbl@readstream}%
3467     \catcode`\[=12 \catcode`\]=12 \catcode`\==12
3468     \catcode`\;=12 \catcode`\|=12 %
3469     \bbl@read@ini{##1}{font and identification data}%
3470     \endinput          % babel- .tex may contain onlypreamble's
3471     \endgroup}%        boxed, to avoid extra spaces:
3472   {\setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}}{}}%

```

Alphabetic counters must be converted from a space separated list to an \ifcase structure.

```

3473 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}
3474   \ifx\#1%           % \ before, in case #1 is multiletter
3475     \bbl@exp{%
3476       \def\#1\bbl@tempa####1{%
3477         \ifcase>####1\space\the\toks@<\else>\@ctrerr<\fi>}}%
3478     \else
3479       \toks@\expandafter{\the\toks@<\or #1>%
3480       \expandafter\bbl@buildifcase
3481     \fi}

```

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

```

3482 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@\language}{#2}}
3483 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3484 \newcommand\localecounter[2]{%
3485   \expandafter\bbl@localecntr\csname c@#2\endcsname{#1}}
3486 \def\bbl@alphnumeral#1#2{%
3487   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3488 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3489   \ifcase\car#8\@nil\or % Currently <10000, but prepared for bigger
3490     \bbl@alphnumeral@ii{#9}000000#1\or
3491     \bbl@alphnumeral@ii{#9}00000#1#2\or
3492     \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3493     \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3494     \bbl@alphnum@invalid{>9999}%
3495   \fi}
3496 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3497   \bbl@ifunset{\bbl@cntr@#1.F.\number#5#6#7#8@\language}%
3498   {\bbl@cs{cntr@#1.4@\language}{#5}
3499     \bbl@cs{cntr@#1.3@\language}{#6}
3500     \bbl@cs{cntr@#1.2@\language}{#7}

```

```

3501 \bbl@cs{cntr@#1.1@\language#8%
3502 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3503 \bbl@ifunset{bbl@cntr@#1.S.321@\language#}%
3504 {\bbl@cs{cntr@#1.S.321@\language#}%
3505 \fi}%
3506 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language#}}
3507 \def\bbl@alphnum@invalid#1{%
3508 \bbl@error{Alphabetic numeral too large (#1)}%
3509 {Currently this is the limit.}}

```

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

```

3510 \newcommand\localeinfo[1]{%
3511 \bbl@ifunset{bbl@\csname bbl@info@#1\endcsname @\language#}%
3512 {\bbl@error{I've found no info for the current locale.\%
3513 The corresponding ini file has not been loaded\%
3514 Perhaps it doesn't exist}%
3515 {See the manual for details.}}%
3516 {\bbl@cs{\csname bbl@info@#1\endcsname @\language#}}
3517 % \@namedef{bbl@info@name.locale}{lname}
3518 \@namedef{bbl@info@tag.ini}{lini}
3519 \@namedef{bbl@info@name.english}{elname}
3520 \@namedef{bbl@info@name.opentype}{lname}
3521 \@namedef{bbl@info@tag.bcp47}{lbcpl}
3522 \@namedef{bbl@info@tag.opentype}{lotf}
3523 \@namedef{bbl@info@script.name}{esname}
3524 \@namedef{bbl@info@script.name.opentype}{sname}
3525 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3526 \@namedef{bbl@info@script.tag.opentype}{sotf}
3527 \let\bbl@ensureinfo@gobble
3528 \newcommand\BabelEnsureInfo{%
3529 \def\bbl@ensureinfo##1{%
3530 \ifx\InputIfFileExists\@undefined\else % not in plain
3531 \bbl@ifunset{bbl@lname@##1}{\bbl@ini@basic{##1}}}%
3532 \fi}}

```

More general, but non-expandable, is `\getlocaleproperty`. To inspect every possible loaded ini, we define `\LocaleForEach`, where `\bbl@ini@loaded` is a comma-separated list of locales, built by `\bbl@read@ini`.

```

3533 \newcommand\getlocaleproperty[3]{%
3534 \let#1\relax
3535 \def\bbl@elt##1##2##3{%
3536 \bbl@ifsamestring{##1/##2}{##3}%
3537 {\providecommand#1{##3}%
3538 \def\bbl@elt####1####2####3{}}%
3539 {}}%
3540 \bbl@cs{inidata@#2}%
3541 \ifx#1\relax
3542 \bbl@error
3543 {Unknown key for locale '#2':\%
3544 #3\%
3545 \string#1 will be set to \relax}%
3546 {Perhaps you misspelled it.}%
3547 \fi}
3548 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```



## 10 Adjusting the Babel bahavior

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

```
3549 \newcommand\babeladjust[1]{% TODO. Error handling.
3550   \bbl@forkv{#1}{%
3551     \bbl@ifunset{bbl@ADJ@##1@##2}%
3552     {\bbl@cs{ADJ@##1}{##2}}%
3553     {\bbl@cs{ADJ@##1@##2}}}
3554 %
3555 \def\bbl@adjust@lua#1#2{%
3556   \ifvmode
3557     \ifnum\currentgrouplevel=\z@
3558       \directlua{ Babel.#2 }%
3559       \expandafter\expandafter\expandafter\@gobble
3560     \fi
3561   \fi
3562   {\bbl@error   % The error is gobbled if everything went ok.
3563     {Currently, #1 related features can be adjusted only\\%
3564       in the main vertical list.}%
3565     {Maybe things change in the future, but this is what it is.}}}
3566 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3567   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3568 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3569   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3570 \@namedef{bbl@ADJ@bidi.text@on}{%
3571   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3572 \@namedef{bbl@ADJ@bidi.text@off}{%
3573   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3574 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3575   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3576 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3577   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3578 %
3579 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3580   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3581 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3582   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3583 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3584   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3585 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3586   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3587 %
3588 \def\bbl@adjust@layout#1{%
3589   \ifvmode
3590     #1%
3591     \expandafter\@gobble
3592   \fi
3593   {\bbl@error   % The error is gobbled if everything went ok.
3594     {Currently, layout related features can be adjusted only\\%
3595       in vertical mode.}%
3596     {Maybe things change in the future, but this is what it is.}}}
3597 \@namedef{bbl@ADJ@layout.tabular@on}{%
3598   \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}}
3599 \@namedef{bbl@ADJ@layout.tabular@off}{%
3600   \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}}
3601 \@namedef{bbl@ADJ@layout.lists@on}{%
3602   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3603 \@namedef{bbl@ADJ@layout.lists@on}{%
```

```

3604 \bbl@adjust@layout{\let\list\bbl@OL@list}}
3605 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
3606 \bbl@activateposthyphen}
3607 %
3608 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3609 \bbl@bcpallowedtrue}
3610 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3611 \bbl@bcpallowedfalse}
3612 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3613 \def\bbl@bcp@prefix{#1}}
3614 \def\bbl@bcp@prefix{bcp47-}
3615 \@namedef{bbl@ADJ@autoload.options}#1{%
3616 \def\bbl@autoload@options{#1}}
3617 \let\bbl@autoload@bcptoptions\@empty
3618 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3619 \def\bbl@autoload@bcptoptions{#1}}
3620 % TODO: use babel name, override
3621 %
3622 % As the final task, load the code for lua.
3623 %
3624 \ifx\directlua\@undefined\else
3625 \ifx\bbl@luapatterns\@undefined
3626 \input luababel.def
3627 \fi
3628 \fi
3629 </core>

```

A proxy file for switch.def

```

3630 <*kernel>
3631 \let\bbl@onlyswitch\@empty
3632 \input babel.def
3633 \let\bbl@onlyswitch\@undefined
3634 </kernel>
3635 <*patterns>

```

## 11 Loading hyphenation patterns

The following code is meant to be read by  $\text{iniT}_{\text{E}}\text{X}$  because it should instruct  $\text{T}_{\text{E}}\text{X}$  to read hyphenation patterns. To this end the `docstrip` option `patterns` can be used to include this code in the file `hyphen.cfg`. Code is written with lower level macros.

To make sure that  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$  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.

```

3636 <<Make sure ProvidesFile is defined>>
3637 \ProvidesFile{hyphen.cfg}[\<date>\<version>] Babel hyphens]
3638 \xdef\bbl@format{\jobname}
3639 \def\bbl@version{\<version>}
3640 \def\bbl@date{\<date>}
3641 \ifx\AtBeginDocument\@undefined
3642 \def\@empty{}
3643 \let\orig@dump\dump
3644 \def\dump{%

```

```

3645 \ifx\@ztryfc\@undefined
3646 \else
3647 \toks0=\expandafter{\@preamblecmds}%
3648 \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
3649 \def\@begindocumenthook{}%
3650 \fi
3651 \let\dump\orig@dump\let\orig@dump\@undefined\dump}
3652 \fi
3653 <<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.

```

3654 \def\process@line#1#2 #3 #4 {%
3655 \ifx=#1%
3656 \process@synonym{#2}%
3657 \else
3658 \process@language{#1#2}{#3}{#4}%
3659 \fi
3660 \ignorespaces}

```

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

```

3661 \toks@{}
3662 \def\bbl@languages{}

```

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

Otherwise the name will be a synonym for the language loaded last.

We also need to copy the `hyphenmin` parameters for the synonym.

```

3663 \def\process@synonym#1{%
3664 \ifnum\last@language=\m@ne
3665 \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
3666 \else
3667 \expandafter\chardef\csname l@#1\endcsname\last@language
3668 \wlog{\string\l@#1=\string\language\the\last@language}%
3669 \expandafter\let\csname #1hyphenmins\endcsname\expandafter\endcsname
3670 \csname\language\hyphenmins\endcsname
3671 \let\bbl@elt\relax
3672 \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}}%
3673 \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.

```

3674 \def\process@language#1#2#3{%
3675   \expandafter\addlanguage\csname l@#1\endcsname
3676   \expandafter\language\csname l@#1\endcsname
3677   \edef\languagename{#1}%
3678   \bbl@hook@everylanguage{#1}%
3679   % > luatex
3680   \bbl@get@enc#1::\@@@
3681   \begingroup
3682     \lefthyphenmin\m@ne
3683     \bbl@hook@loadpatterns{#2}%
3684     % > luatex
3685     \ifnum\lefthyphenmin=\m@ne
3686     \else
3687       \expandafter\xdef\csname #1hyphenmins\endcsname{%
3688         \the\lefthyphenmin\the\righthyphenmin}%
3689     \fi
3690   \endgroup
3691   \def\bbl@tempa{#3}%
3692   \ifx\bbl@tempa\@empty\else
3693     \bbl@hook@loadexceptions{#3}%
3694     % > luatex
3695   \fi
3696   \let\bbl@elt\relax
3697   \edef\bbl@languages{%
3698     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
3699   \ifnum\the\language=\z@
3700     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
3701       \set@hyphenmins\tw@\thr@@\relax
3702     \else
3703       \expandafter\expandafter\expandafter\set@hyphenmins
3704       \csname #1hyphenmins\endcsname
3705     \fi
3706     \the\toks@
3707     \toks@{}%
3708   \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.

```

3709 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}

```

Now, hooks are defined. For efficiency reasons, they are dealt here in a special way. Besides luatex, format-specific configuration files are taken into account. loadkernel currently loads nothing, but define some basic macros instead.

```

3710 \def\bbbl@hook@everylanguage#1{}
3711 \def\bbbl@hook@loadpatterns#1{\input #1\relax}
3712 \let\bbbl@hook@loadexceptions\bbbl@hook@loadpatterns
3713 \def\bbbl@hook@loadkernel#1{%
3714   \def\addlanguage{\alloc@9\language\chardef\ccclvi}%
3715   \def\adddialect##1##2{%
3716     \global\chardef##1##2\relax
3717     \wlog{\string##1 = a dialect from \string\language##2}}%
3718   \def\iflanguage##1{%
3719     \expandafter\ifx\csname l@##1\endcsname\relax
3720       \nolannerr{##1}%
3721     \else
3722       \ifnum\csname l@##1\endcsname=\language
3723         \expandafter\expandafter\expandafter\@firstoftwo
3724       \else
3725         \expandafter\expandafter\expandafter\@secondoftwo
3726       \fi
3727     \fi}%
3728   \def\providehyphenmins##1##2{%
3729     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
3730       \namedef{##1hyphenmins}{##2}%
3731     \fi}%
3732   \def\set@hyphenmins##1##2{%
3733     \lefthyphenmin##1\relax
3734     \righthyphenmin##2\relax}%
3735   \def\selectlanguage{%
3736     \errhelp{Selecting a language requires a package supporting it}%
3737     \errmessage{Not loaded}}%
3738   \let\foreignlanguage\selectlanguage
3739   \let\otherlanguage\selectlanguage
3740   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
3741   \def\bbbl@usehooks##1##2{% TODO. Temporary!!
3742     \def\setlocale{%
3743       \errhelp{Find an armchair, sit down and wait}%
3744       \errmessage{Not yet available}}%
3745     \let\uselocale\setlocale
3746     \let\locale\setlocale
3747     \let\selectlocale\setlocale
3748     \let\localename\setlocale
3749     \let\textlocale\setlocale
3750     \let\textlanguage\setlocale
3751     \let\languagetext\setlocale}
3752   \begingroup
3753     \def\AddBabelHook#1#2{%
3754       \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
3755         \def\next{\toks1}%
3756       \else
3757         \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
3758       \fi
3759     \next}
3760   \ifx\directlua\@undefined
3761     \ifx\XeTeXinputencoding\@undefined\else
3762       \input xebabel.def
3763     \fi
3764   \else

```

```

3765 \input luababel.def
3766 \fi
3767 \openin1 = babel-\bbl@format.cfg
3768 \ifeof1
3769 \else
3770 \input babel-\bbl@format.cfg\relax
3771 \fi
3772 \closein1
3773 \endgroup
3774 \bbl@hook@loadkernel{switch.def}

```

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

```

3775 \openin1 = language.dat

```

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

```

3776 \def\languagename{english}%
3777 \ifeof1
3778 \message{I couldn't find the file language.dat,\space
3779         I will try the file hyphen.tex}
3780 \input hyphen.tex\relax
3781 \chardef\l@english\z@
3782 \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`.

```

3783 \last@language\m@ne

```

We now read lines from the file until the end is found. While reading from the input, it is useful to switch off recognition of the end-of-line character. This saves us stripping off spaces from the contents of the control sequence.

```

3784 \loop
3785 \endlinechar\m@ne
3786 \read1 to \bbl@line
3787 \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.

```

3788 \if T\ifeof1F\fi T\relax
3789 \ifx\bbl@line\empty\else
3790 \edef\bbl@line{\bbl@line\space\space\space}%
3791 \expandafter\process@line\bbl@line\relax
3792 \fi
3793 \repeat

```

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

```

3794 \begingroup
3795 \def\bbl@elt#1#2#3#4{%
3796 \global\language=#2\relax
3797 \gdef\languagename{#1}%
3798 \def\bbl@elt##1##2##3##4{}}%
3799 \bbl@languages
3800 \endgroup
3801 \fi
3802 \closein1

```

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

```
3803 \if/\the\toks@/\else
3804 \errhelp{language.dat loads no language, only synonyms}
3805 \errmessage{Orphan language synonym}
3806 \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.

```
3807 \let\bbl@line\@undefined
3808 \let\process@line\@undefined
3809 \let\process@synonym\@undefined
3810 \let\process@language\@undefined
3811 \let\bbl@get@enc\@undefined
3812 \let\bbl@hyph@enc\@undefined
3813 \let\bbl@tempa\@undefined
3814 \let\bbl@hook@loadkernel\@undefined
3815 \let\bbl@hook@everylanguage\@undefined
3816 \let\bbl@hook@loadpatterns\@undefined
3817 \let\bbl@hook@loadexceptions\@undefined
3818 \let\patterns\@undefined
```

Here the code for iniT<sub>E</sub>X ends.

## 12 Font handling with fontspec

Add the bidi handler just before luaotfload, which is loaded by default by LaTeX. Just in case, consider the possibility it has not been loaded. First, a couple of definitions related to bidi [misplaced].

```
3819 <<(*More package options)>> ≡
3820 \ifodd\bbl@engine
3821 \DeclareOption{bidi=basic-r}%
3822 {\ExecuteOptions{bidi=basic}}
3823 \DeclareOption{bidi=basic}%
3824 {\let\bbl@beforeforeign\leavevmode
3825 % TODO - to locale_props, not as separate attribute
3826 \newattribute\bbl@attr@dir
3827 % I don't like it, hackish:
3828 \frozen@everymath\expandafter{%
3829 \expandafter\bbl@mathboxdir\the\frozen@everymath}%
3830 \frozen@everydisplay\expandafter{%
3831 \expandafter\bbl@mathboxdir\the\frozen@everydisplay}%
3832 \bbl@exp{\output{\bodydir\pagedir\the\output}}}%
3833 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}}
3834 \else
3835 \DeclareOption{bidi=basic-r}%
3836 {\ExecuteOptions{bidi=basic}}
3837 \DeclareOption{bidi=basic}%
3838 {\bbl@error
3839 {The bidi method 'basic' is available only in\%
3840 luatex. I'll continue with 'bidi=default', so\%
3841 expect wrong results}%
3842 {See the manual for further details.}%
3843 \let\bbl@beforeforeign\leavevmode
3844 \AtEndOfPackage{%
3845 \EnableBabelHook{babel-bidi}%
```

```

3846     \bbl@xebidipar}}
3847 \def\bbl@loadxebidi#1{%
3848   \ifx\RTLfootnotetext\@undefined
3849     \AtEndOfPackage{%
3850       \EnableBabelHook{babel-bidi}%
3851       \ifx\fontspec\@undefined
3852         \usepackage{fontspec}% bidi needs fontspec
3853       \fi
3854       \usepackage#1{bidi}}%
3855     \fi}
3856 \DeclareOption{bidi=bidi}%
3857   {\bbl@tentative{bidi=bidi}%
3858     \bbl@loadxebidi{}}
3859 \DeclareOption{bidi=bidi-r}%
3860   {\bbl@tentative{bidi=bidi-r}%
3861     \bbl@loadxebidi{[rldocument]}}
3862 \DeclareOption{bidi=bidi-l}%
3863   {\bbl@tentative{bidi=bidi-l}%
3864     \bbl@loadxebidi{}}
3865 \fi
3866 \DeclareOption{bidi=default}%
3867   {\let\bbl@beforeforeign\leavevmode
3868     \ifodd\bbl@engine
3869       \newattribute\bbl@attr@dir
3870       \bbl@exp{\output{\bodydir\pagedir\the\output}}%
3871     \fi
3872     \AtEndOfPackage{%
3873       \EnableBabelHook{babel-bidi}%
3874       \ifodd\bbl@engine\else
3875         \bbl@xebidipar
3876       \fi}}
3877 <</More package options>>

```

With explicit languages, we could define the font at once, but we don't. Just wait and see if the language is actually activated. `\bbl@font` replaces hardcoded font names inside `\. . family` by the corresponding macro `\. . default`.

```

3878 <<(*Font selection)>> ≡
3879 \bbl@trace{Font handling with fontspec}
3880 \@onlypreamble\babelfont
3881 \newcommand\babelfont[2][1]{% 1=langs/scripts 2=fam
3882   \bbl@foreach{#1}{%
3883     \expandafter\ifx\csname date##1\endcsname\relax
3884       \IfFileExists{babel-##1.tex}%
3885         {\babelprovide{##1}}%
3886       {}%
3887     \fi}%
3888 \edef\bbl@tempa{#1}%
3889 \def\bbl@tempb{#2}% Used by \bbl@bblfont
3890 \ifx\fontspec\@undefined
3891   \usepackage{fontspec}%
3892 \fi
3893 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
3894 \bbl@bblfont}
3895 \newcommand\bbl@bblfont[2][1]{% 1=features 2=fontname, @font=rm|sf|tt
3896   \bbl@ifunset{\bbl@tempb family}%
3897     {\bbl@providefam{\bbl@tempb}}%
3898     {\bbl@exp{%
3899       \\bbl@sreplace\<\bbl@tempb family >%
3900       {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%

```



```

3901 % For the default font, just in case:
3902 \bbl@ifunset{\bbl@lsys@\language}\bbl@provide@lsys{\language}}{}%
3903 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
3904 {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}}% save \bbl@rmdflt@
3905 \bbl@exp{%
3906 \let<\bbl@tempb dflt@\language>\<\bbl@tempb dflt@>%
3907 \\\bbl@font@set<\bbl@tempb dflt@\language>%
3908 \<\bbl@tempb default>\<\bbl@tempb family>}}%
3909 {\bbl@foreach\bbl@tempa{% ie \bbl@rmdflt@lang / *scrt
3910 \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:

```

3911 \def\bbl@providefam#1{%
3912 \bbl@exp{%
3913 \\\newcommand<#1default>{}% Just define it
3914 \\\bbl@add@list\\bbl@font@fams{#1}%
3915 \\\DeclareRobustCommand<#1family>{%
3916 \\\not@math@alphabet<#1family>\relax
3917 \\\fontfamily<#1default>\selectfont}%
3918 \\\DeclareTextFontCommand{\<text#1>}{<#1family>}}

```

The following macro is activated when the hook `babel-fontspec` is enabled. But before we define a macro for a warning, which sets a flag to avoid duplicate them.

```

3919 \def\bbl@nostdfont#1{%
3920 \bbl@ifunset{\bbl@WFF@f@family}%
3921 {\bbl@csarg\gdef{\bbl@WFF@f@family}{% Flag, to avoid dupl warns
3922 \bbl@infowarn{The current font is not a babel standard family:\%
3923 #1%
3924 \fontname\font\\%
3925 There is nothing intrinsically wrong with this warning, and\\%
3926 you can ignore it altogether if you do not need these\\%
3927 families. But if they are used in the document, you should be\\%
3928 aware 'babel' will no set Script and Language for them, so\\%
3929 you may consider defining a new family with \string\babelfont.\\%
3930 See the manual for further details about \string\babelfont.\\%
3931 Reported}}
3932 {}}%
3933 \gdef\bbl@switchfont{%
3934 \bbl@ifunset{\bbl@lsys@\language}\bbl@provide@lsys{\language}}{}%
3935 \bbl@exp{% eg Arabic -> arabic
3936 \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}%
3937 \bbl@foreach\bbl@font@fams{%
3938 \bbl@ifunset{\bbl@##1dflt@\language}% (1) language?
3939 {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}% (2) from script?
3940 {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
3941 {}% 123=F - nothing!
3942 {\bbl@exp{% 3=T - from generic
3943 \global\let<\bbl@##1dflt@\language>%
3944 \<\bbl@##1dflt@>}}}%
3945 {\bbl@exp{% 2=T - from script
3946 \global\let<\bbl@##1dflt@\language>%
3947 \<\bbl@##1dflt@*\bbl@tempa>}}}%
3948 {}% 1=T - language, already defined
3949 \def\bbl@tempa{\bbl@nostdfont}}%
3950 \bbl@foreach\bbl@font@fams{% don't gather with prev for
3951 \bbl@ifunset{\bbl@##1dflt@\language}%
3952 {\bbl@cs{famrst@##1}%
3953 \global\bbl@csarg\let{famrst@##1}\relax}%
3954 {\bbl@exp{% order is relevant

```

```

3955      \\bbl@add\\originalTeX{%
3956      \\bbl@font@rst{\\bbl@cl{##1dflt}}}%
3957      \\<##1default>\\<##1family>{##1}}}%
3958      \\bbl@font@set\\<bbl@##1dflt@\\languagename>% the main part!
3959      \\<##1default>\\<##1family>}}}%
3960      \\bbl@ifrestoring{}{\\bbl@tempa}}}%

```

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

```

3961 \\ifx\\f@family\\undefined\\else % if latex
3962 \\ifcase\\bbl@engine % if pdftex
3963 \\let\\bbl@ckeckstdfonts\\relax
3964 \\else
3965 \\def\\bbl@ckeckstdfonts{%
3966 \\begingroup
3967 \\global\\let\\bbl@ckeckstdfonts\\relax
3968 \\let\\bbl@tempa\\empty
3969 \\bbl@foreach\\bbl@font@fams{%
3970 \\bbl@ifunset{\\bbl@##1dflt@}%
3971 {\\nameuse{##1family}}%
3972 \\bbl@csarg\\gdef{WFF@\\f@family}}}% Flag
3973 \\bbl@exp{\\bbl@add\\bbl@tempa{* \\<##1family>= \\f@family\\}%
3974 \\space\\space\\fontname\\font\\}%
3975 \\bbl@csarg\\xdef{##1dflt@}{\\f@family}%
3976 \\expandafter\\xdef\\csname ##1default\\endcsname{\\f@family}}}%
3977 {}}%
3978 \\ifx\\bbl@tempa\\empty\\else
3979 \\bbl@infowarn{The following font families will use the default\\%
3980 settings for all or some languages:\\%
3981 \\bbl@tempa
3982 There is nothing intrinsically wrong with it, but\\%
3983 'babel' will no set Script and Language, which could\\%
3984 be relevant in some languages. If your document uses\\%
3985 these families, consider redefining them with \\string\\babelfont.\\%
3986 Reported}%
3987 \\fi
3988 \\endgroup}
3989 \\fi
3990 \\fi

```

Now the macros defining the font with `fontspec`.

When there are repeated keys in `fontspec`, the last value wins. So, we just place the ini settings at the beginning, and user settings will take precedence. We must deactivate temporarily `\bbl@mapselect` because `\selectfont` is called internally when a font is defined.

```

3991 \\def\\bbl@font@set#1#2#3{% eg \\bbl@rmdflt@lang \\rmdefault \\rmfamily
3992 \\bbl@xin@{<>}{#1}}%
3993 \\ifin@
3994 \\bbl@exp{\\bbl@fontspec@set\\#1\\expandafter\\gobbletwo#1\\#3}%
3995 \\fi
3996 \\bbl@exp{%
3997 \\def\\#2{#1}% eg, \\rmdefault{\\bbl@rmdflt@lang}
3998 \\bbl@ifsamestring{#2}{\\f@family}{\\#3\\let\\bbl@tempa\\relax}}}%
3999 % TODO - next should be global?, but even local does its job. I'm
4000 % still not sure -- must investigate:
4001 \\def\\bbl@fontspec@set#1#2#3#4{% eg \\bbl@rmdflt@lang fnt-opt fnt-nme \\xxfamily
4002 \\let\\bbl@tempe\\bbl@mapselect
4003 \\let\\bbl@mapselect\\relax
4004 \\let\\bbl@temp@fam#4% eg, '\\rmfamily', to be restored below

```

```

4005 \let#4@empty      %      Make sure \renewfontfamily is valid
4006 \bbl@exp{%
4007   \let\\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
4008   \<keys_if_exist:nnF>{\fontspec-opentype}{Script/\bbl@cl{sname}}%
4009   {\\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4010   \<keys_if_exist:nnF>{\fontspec-opentype}{Language/\bbl@cl{lname}}%
4011   {\\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4012   \\renewfontfamily\\#4%
4013   [\bbl@cs{lsys@\language}\language, #2]}{#3}% ie \bbl@exp{.}{#3}
4014 \begingroup
4015   #4%
4016   \xdef#1{\f@family}%      eg, \bbl@rmdflt@lang{FreeSerif(0)}
4017 \endgroup
4018 \let#4\bbl@temp@fam
4019 \bbl@exp{\let<\bbl@stripslash#4\space>}\bbl@temp@pfam
4020 \let\bbl@mapselect\bbl@tempe}%

```

font@rst and famrst are only used when there is no global settings, to save and restore de previous families. Not really necessary, but done for optimization.

```

4021 \def\bbl@font@rst#1#2#3#4{%
4022   \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.

```

4023 \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 :-).

```

4024 \newcommand\babelFSstore[2][{%
4025   \bbl@ifblank{#1}%
4026   {\bbl@csarg\def{sname@#2}{Latin}}%
4027   {\bbl@csarg\def{sname@#2}{#1}}%
4028   \bbl@provide@dirs{#2}%
4029   \bbl@csarg\ifnum{wdir@#2}>\z@
4030     \let\bbl@beforeforeign\leavevmode
4031     \EnableBabelHook{babel-bidi}%
4032   \fi
4033   \bbl@foreach{#2}{%
4034     \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4035     \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4036     \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4037 \def\bbl@FSstore#1#2#3#4{%
4038   \bbl@csarg\edef{#2default#1}{#3}%
4039   \expandafter\addto\csname extras#1\endcsname{%
4040     \let#4#3%
4041     \ifx#3\f@family
4042       \edef#3{\csname bbl@#2default#1\endcsname}%
4043       \fontfamily{#3}\selectfont
4044     \else
4045       \edef#3{\csname bbl@#2default#1\endcsname}%
4046     \fi}%
4047   \expandafter\addto\csname noextras#1\endcsname{%
4048     \ifx#3\f@family
4049       \fontfamily{#4}\selectfont
4050     \fi
4051     \let#3#4}}
4052 \let\bbl@langfeatures@empty
4053 \def\babelFSfeatures{% make sure \fontspec is redefined once

```

```

4054 \let\bbl@ori@fontspec\fontspec
4055 \renewcommand\fontspec[1][{%
4056   \bbl@ori@fontspec[\bbl@langfeatures##1]}
4057 \let\babelFSfeatures\bbl@FSfeatures
4058 \babelFSfeatures}
4059 \def\bbl@FSfeatures#1#2{%
4060   \expandafter\addto\csname extras#1\endcsname{%
4061     \babel@save\bbl@langfeatures
4062     \edef\bbl@langfeatures{#2,}}
4063 \</Font selection>

```

## 13 Hooks for XeTeX and LuaTeX

### 13.1 XeTeX

Unfortunately, the current encoding cannot be retrieved and therefore it is reset always to `utf8`, which seems a sensible default.

```

4064 \<{*Footnote changes}> ≡
4065 \bbl@trace{Bidi footnotes}
4066 \ifx\bbl@beforeforeign\leavevmode
4067   \def\bbl@footnote#1#2#3{%
4068     \@ifnextchar[%
4069       {\bbl@footnote@o{#1}{#2}{#3}}%
4070       {\bbl@footnote@x{#1}{#2}{#3}}}
4071   \def\bbl@footnote@x#1#2#3#4{%
4072     \bgroup
4073     \select@language@x{\bbl@main@language}%
4074     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4075     \egroup}
4076   \def\bbl@footnote@o#1#2#3[#4]#5{%
4077     \bgroup
4078     \select@language@x{\bbl@main@language}%
4079     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4080     \egroup}
4081   \def\bbl@footnotetext#1#2#3{%
4082     \@ifnextchar[%
4083       {\bbl@footnotetext@o{#1}{#2}{#3}}%
4084       {\bbl@footnotetext@x{#1}{#2}{#3}}}
4085   \def\bbl@footnotetext@x#1#2#3#4{%
4086     \bgroup
4087     \select@language@x{\bbl@main@language}%
4088     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4089     \egroup}
4090   \def\bbl@footnotetext@o#1#2#3[#4]#5{%
4091     \bgroup
4092     \select@language@x{\bbl@main@language}%
4093     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4094     \egroup}
4095   \def\BabelFootnote#1#2#3#4{%
4096     \ifx\bbl@fn@footnote\undefined
4097       \let\bbl@fn@footnote\footnote
4098     \fi
4099     \ifx\bbl@fn@footnotetext\undefined
4100       \let\bbl@fn@footnotetext\footnotetext
4101     \fi
4102     \bbl@ifblank{#2}%
4103     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}}

```

```

4104 \namedef{\bbl@stripslash#1text}%
4105 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4106 {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}}%
4107 \namedef{\bbl@stripslash#1text}%
4108 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4109 \fi
4110 <</Footnote changes>>

```

Now, the code.

```

4111 (*xetex)
4112 \def\BabelStringsDefault{unicode}
4113 \let\xebbl@stop\relax
4114 \AddBabelHook{xetex}{encodedcommands}{%
4115 \def\bbl@tempa{#1}%
4116 \ifx\bbl@tempa@empty
4117 \XeTeXinputencoding"bytes"%
4118 \else
4119 \XeTeXinputencoding"#1"%
4120 \fi
4121 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4122 \AddBabelHook{xetex}{stopcommands}{%
4123 \xebbl@stop
4124 \let\xebbl@stop\relax}
4125 \def\bbl@intraspace#1 #2 #3\@@{%
4126 \bbl@csarg\gdef\xeisp@{language}%
4127 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4128 \def\bbl@intrapenalty#1\@@{%
4129 \bbl@csarg\gdef\xeipn@{language}%
4130 {\XeTeXlinebreakpenalty #1\relax}}
4131 \def\bbl@provide@intraspace{%
4132 \bbl@xin@{\bbl@cl{lnbrk}}{s}%
4133 \ifin@else\bbl@xin@{\bbl@cl{lnbrk}}{c}\fi
4134 \ifin@
4135 \bbl@ifunset{\bbl@intsp@{language}}{%
4136 {\expandafter\ifx\cname\bbl@intsp@{language}\endcsname@empty\else
4137 \ifx\bbl@KVP@intraspace@nil
4138 \bbl@exp{%
4139 \bbl@intraspace\bbl@cl{intsp}\@@}%
4140 \fi
4141 \ifx\bbl@KVP@intrapenalty@nil
4142 \bbl@intrapenalty0\@@
4143 \fi
4144 \fi
4145 \ifx\bbl@KVP@intraspace@nil\else % We may override the ini
4146 \expandafter\bbl@intraspace\bbl@KVP@intraspace@@
4147 \fi
4148 \ifx\bbl@KVP@intrapenalty@nil\else
4149 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty@@
4150 \fi
4151 \bbl@exp{%
4152 \bbl@add{<extras\language>{%
4153 \XeTeXlinebreaklocale "\bbl@cl{lbcpr}"%
4154 \<bbl@xeisp@{language}>%
4155 \<bbl@xeipn@{language}>%
4156 \bbl@toglobal{<extras\language>%
4157 \bbl@add{<noextras\language>{%
4158 \XeTeXlinebreaklocale "en"%
4159 \bbl@toglobal{<noextras\language>}}%
4160 \ifx\bbl@ispace@undefined

```

```

4161      \gdef\bbl@ispace size{\bbl@cl{x eisp}}}%
4162      \ifx\AtBeginDocument\@notprerr
4163        \expandafter\@secondoftwo % to execute right now
4164        \fi
4165        \AtBeginDocument{%
4166          \expandafter\bbl@add
4167          \csname selectfont \endcsname{\bbl@ispace size}%
4168          \expandafter\bbl@tglobal\csname selectfont \endcsname}%
4169        \fi}%
4170    \fi}
4171 \ifx\DisableBabelHook\@undefined\endinput\fi
4172 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4173 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4174 \DisableBabelHook{babel-fontspec}
4175 <<Font selection>>
4176 \input txtbabel.def
4177 </xetex>

```

## 13.2 Layout

*In progress.*

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titleps, and geometry.

\bbl@startskip and \bbl@endskip are available to package authors. Thanks to the  $\TeX$  expansion mechanism the following constructs are valid: \adim\bbl@startskip, \advance\bbl@startskip\adim, \bbl@startskip\adim.

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

```

4178 <*texxet>
4179 \providecommand\bbl@provide@intraspace{}
4180 \bbl@trace{Redefinitions for bidi layout}
4181 \def\bbl@sspre@caption{%
4182   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir}\bbl@main@language}}}%
4183   \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4184   \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4185   \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4186   \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4187     \def\@hangfrom#1{%
4188       \setbox\@tempboxa\hbox{#1}%
4189       \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4190       \noindent\box\@tempboxa}
4191   \def\raggedright{%
4192     \let\@centercr
4193     \bbl@startskip\z@skip
4194     \@rightskip\@flushglue
4195     \bbl@endskip\@rightskip
4196     \parindent\z@
4197     \parfillskip\bbl@startskip}
4198   \def\raggedleft{%
4199     \let\@centercr
4200     \bbl@startskip\@flushglue
4201     \bbl@endskip\z@skip
4202     \parindent\z@
4203     \parfillskip\bbl@endskip}
4204   \fi
4205   \IfBabelLayout{lists}
4206   {\bbl@sreplace\list

```

```

4207     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4208 \def\bbl@listleftmargin{%
4209   \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4210 \ifcase\bbl@engine
4211   \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4212   \def\p@enumiii{\p@enumii}\theenumii{}\fi
4213 \fi
4214 \bbl@sreplace\@verbatim
4215   {\leftskip\@totalleftmargin}%
4216   {\bbl@startskip\textwidth
4217     \advance\bbl@startskip-\linewidth}%
4218 \bbl@sreplace\@verbatim
4219   {\rightskip\z@skip}%
4220   {\bbl@endskip\z@skip}}%
4221 {}
4222 \IfBabelLayout{contents}
4223   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4224     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4225 {}
4226 \IfBabelLayout{columns}
4227   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4228     \def\bbl@outputbox#1{%
4229       \hb@xt@\textwidth{%
4230         \hskip\columnwidth
4231         \hfil
4232         {\normalcolor\vrule \@width\columnseprule}%
4233         \hfil
4234         \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4235         \hskip-\textwidth
4236         \hb@xt@\columnwidth{\box\@outputbox \hss}%
4237         \hskip\columnsep
4238         \hskip\columnwidth}}}%
4239 {}
4240 <<Footnote changes>>
4241 \IfBabelLayout{footnotes}%
4242   {\BabelFootnote\footnote\languagename{}\{}}%
4243   \BabelFootnote\localfootnote\languagename{}\{}}%
4244   \BabelFootnote\mainfootnote{}\{}}%
4245 {}

```

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

```

4246 \IfBabelLayout{counters}%
4247   {\let\bbl@latinarabic=\@arabic
4248     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4249     \let\bbl@asciroman=\@roman
4250     \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4251     \let\bbl@asciiRoman=\@Roman
4252     \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}%
4253 </texet>

```

### 13.3 LuaTeX

The loader for `luatex` is based solely on `language.dat`, which is read on the fly. The code shouldn't be executed when the format is build, so we check if `\AddBabelHook` is defined. Then comes a modified version of the loader in `hyphen.cfg` (without the `hyphenmins` stuff, which is under the direct control of `babel`).

The names `\l@<language>` are defined and take some value from the beginning because all ldf files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first selected (which usually means when the ldf finishes). If a language has been loaded, `\bbl@hyphendata@<num>` exists (with the names of the files read).

The default setup preloads the first language into the format. This is intended mainly for ‘english’, so that it’s available without further intervention from the user. To avoid duplicating it, the following rule applies: if the “0th” language and the first language in `language.dat` have the same name then just ignore the latter. If there are new synonymous, they are added, but note if the language patterns have not been preloaded they won’t at run time.

Other preloaded languages could be read twice, if they have been preloaded into the format. This is not optimal, but it shouldn’t happen very often – with luatex patterns are best loaded when the document is typeset, and the “0th” language is preloaded just for backwards compatibility.

As of 1.1b, lua(e)tex is taken into account. Formerly, loading of patterns on the fly didn’t work in this format, but with the new loader it does. Unfortunately, the format is not based on babel, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format `language.dat` is used (under the principle of a single source), instead of `language.def`.

Of course, there is room for improvements, like tools to read and reassign languages, which would require modifying the language list, and better error handling.

We need catcode tables, but no format (targeted by babel) provide a command to allocate them (although there are packages like `ctablestack`). FIX - This isn’t true anymore. For the moment, a dangerous approach is used - just allocate a high random number and cross the fingers. To complicate things, `etex.sty` changes the way languages are allocated.

This files is read at three places: (1) when `plain.def`, `babel.sty` starts, to read the list of available languages from `language.dat` (for the base option); (2) at `hyphen.cfg`, to modify some macros; (3) in the middle of `plain.def` and `babel.sty`, by `babel.def`, with the commands and other definitions for luatex (eg, `\babelpatterns`).

```

4254 (*luatex)
4255 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4256 \bbl@trace{Read language.dat}
4257 \ifx\bbl@readstream\undefined
4258   \csname newread\endcsname\bbl@readstream
4259 \fi
4260 \begingroup
4261   \toks@{}
4262   \count@ \z@ % 0=start, 1=0th, 2=normal
4263   \def\bbl@process@line#1#2 #3 #4 {%
4264     \ifx=#1%
4265       \bbl@process@synonym{#2}%
4266     \else
4267       \bbl@process@language{#1#2}{#3}{#4}%
4268     \fi
4269     \ignorespaces}
4270   \def\bbl@manylang{%
4271     \ifnum\bbl@last>\@ne
4272       \bbl@info{Non-standard hyphenation setup}%
4273     \fi
4274     \let\bbl@manylang\relax}
4275   \def\bbl@process@language#1#2#3{%
4276     \ifcase\count@
4277       \@ifundefined{zth#1}{\count@ \tw@}{\count@ \@ne}%
4278     \or
4279       \count@ \tw@

```



```

4280 \fi
4281 \ifnum\count@=\tw@
4282 \expandafter\addlanguage\csname l@#1\endcsname
4283 \language\allocationnumber
4284 \chardef\bb1@last\allocationnumber
4285 \bb1@manylang
4286 \let\bb1@elt\relax
4287 \xdef\bb1@languages{%
4288 \bb1@languages\bb1@elt{#1}{\the\language}{#2}{#3}}%
4289 \fi
4290 \the\toks@
4291 \toks@{}}
4292 \def\bb1@process@synonym@aux#1#2{%
4293 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4294 \let\bb1@elt\relax
4295 \xdef\bb1@languages{%
4296 \bb1@languages\bb1@elt{#1}{#2}{}}}%
4297 \def\bb1@process@synonym#1{%
4298 \ifcase\count@
4299 \toks@\expandafter{\the\toks@\relax\bb1@process@synonym{#1}}%
4300 \or
4301 \@ifundefined{zth#1}{\bb1@process@synonym@aux{#1}{}}}%
4302 \else
4303 \bb1@process@synonym@aux{#1}{\the\bb1@last}%
4304 \fi}
4305 \ifx\bb1@languages\@undefined % Just a (sensible?) guess
4306 \chardef\l@english\z@
4307 \chardef\l@USenglish\z@
4308 \chardef\bb1@last\z@
4309 \global\@namedef{\bb1@hyphendata@0}{\hyphen.tex}}
4310 \gdef\bb1@languages{%
4311 \bb1@elt{english}{0}{\hyphen.tex}}%
4312 \bb1@elt{USenglish}{0}{}}
4313 \else
4314 \global\let\bb1@languages@format\bb1@languages
4315 \def\bb1@elt#1#2#3#4{% Remove all except language 0
4316 \ifnum#2>\z@\else
4317 \noexpand\bb1@elt{#1}{#2}{#3}{#4}%
4318 \fi}%
4319 \xdef\bb1@languages{\bb1@languages}%
4320 \fi
4321 \def\bb1@elt#1#2#3#4{\@namedef{zth#1}{}} % Define flags
4322 \bb1@languages
4323 \openin\bb1@readstream=language.dat
4324 \ifeof\bb1@readstream
4325 \bb1@warning{I couldn't find language.dat. No additional\\%
4326 patterns loaded. Reported}%
4327 \else
4328 \loop
4329 \endlinechar\m@ne
4330 \read\bb1@readstream to \bb1@line
4331 \endlinechar\^^M
4332 \if T\ifeof\bb1@readstream F\fi T\relax
4333 \ifx\bb1@line\empty\else
4334 \edef\bb1@line{\bb1@line\space\space\space}%
4335 \expandafter\bb1@process@line\bb1@line\relax
4336 \fi
4337 \repeat
4338 \fi

```

```

4339 \endgroup
4340 \bbl@trace{Macros for reading patterns files}
4341 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
4342 \ifx\babelcatcodetablenum\undefined
4343 \ifx\newcatcodetable\undefined
4344 \def\babelcatcodetablenum{5211}
4345 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4346 \else
4347 \newcatcodetable\babelcatcodetablenum
4348 \newcatcodetable\bbl@pattcodes
4349 \fi
4350 \else
4351 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4352 \fi
4353 \def\bbl@luapatterns#1#2{%
4354 \bbl@get@enc#1::@@@
4355 \setbox\z@\hbox\bgroup
4356 \begingroup
4357 \savecatcodetable\babelcatcodetablenum\relax
4358 \initcatcodetable\bbl@pattcodes\relax
4359 \catcodetable\bbl@pattcodes\relax
4360 \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4361 \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\-=13
4362 \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4363 \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4364 \catcode`\-=12 \catcode`\/=12 \catcode`\|=12 \catcode`\|=12
4365 \catcode`\`=12 \catcode`\`=12 \catcode`\`=12
4366 \input #1\relax
4367 \catcodetable\babelcatcodetablenum\relax
4368 \endgroup
4369 \def\bbl@tempa{#2}%
4370 \ifx\bbl@tempa\empty\else
4371 \input #2\relax
4372 \fi
4373 \egroup}%
4374 \def\bbl@patterns@lua#1{%
4375 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4376 \csname l@#1\endcsname
4377 \edef\bbl@tempa{#1}%
4378 \else
4379 \csname l@#1:\f@encoding\endcsname
4380 \edef\bbl@tempa{#1:\f@encoding}%
4381 \fi\relax
4382 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4383 \@ifundefined{bbl@hyphendata@the\language}%
4384 {\def\bbl@elt##1##2##3##4{%
4385 \ifnum##2=\csname l@#1:\f@encoding\endcsname % #2=spanish, dutch:OT1...
4386 \def\bbl@tempb{##3}%
4387 \ifx\bbl@tempb\empty\else % if not a synonymous
4388 \def\bbl@tempc{##3}{##4}}%
4389 \fi
4390 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4391 \fi}%
4392 \bbl@languages
4393 \@ifundefined{bbl@hyphendata@the\language}%
4394 {\bbl@info{No hyphenation patterns were set for\%
4395 language '\bbl@tempa'. Reported}}%
4396 {\expandafter\expandafter\expandafter\bbl@luapatterns
4397 \csname bbl@hyphendata@the\language\endcsname}}}%

```

```

4398 \endinput\fi
4399 % Here ends \ifx\AddBabelHook\@undefined
4400 % A few lines are only read by hyphen.cfg
4401 \ifx\DisableBabelHook\@undefined
4402   \AddBabelHook{luatex}{everylanguage}{%
4403     \def\process@language##1##2##3{%
4404       \def\process@line####1####2 ####3 ####4 {}}}
4405   \AddBabelHook{luatex}{loadpatterns}{%
4406     \input #1\relax
4407     \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4408       {#{1}{}}}
4409   \AddBabelHook{luatex}{loadexceptions}{%
4410     \input #1\relax
4411     \def\bbl@tempb##1##2{#{#1}{#1}}%
4412     \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
4413       {\expandafter\expandafter\expandafter\bbl@tempb
4414         \csname bbl@hyphendata@the\language\endcsname}}
4415 \endinput\fi
4416 % Here stops reading code for hyphen.cfg
4417 % The following is read the 2nd time it's loaded
4418 \begingroup
4419 \catcode`\%=12
4420 \catcode`\'=12
4421 \catcode`\%=12
4422 \catcode`\:=12
4423 \directlua{
4424   Babel = Babel or {}
4425   function Babel.bytes(line)
4426     return line:gsub(".",
4427       function (chr) return unicode.utf8.char(string.byte(chr)) end)
4428   end
4429   function Babel.begin_process_input()
4430     if luatexbase and luatexbase.add_to_callback then
4431       luatexbase.add_to_callback('process_input_buffer',
4432         Babel.bytes, 'Babel.bytes')
4433     else
4434       Babel.callback = callback.find('process_input_buffer')
4435       callback.register('process_input_buffer', Babel.bytes)
4436     end
4437   end
4438   function Babel.end_process_input ()
4439     if luatexbase and luatexbase.remove_from_callback then
4440       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
4441     else
4442       callback.register('process_input_buffer', Babel.callback)
4443     end
4444   end
4445   function Babel.addpatterns(pp, lg)
4446     local lg = lang.new(lg)
4447     local pats = lang.patterns(lg) or ''
4448     lang.clear_patterns(lg)
4449     for p in pp:gmatch('[^%s]+') do
4450       ss = ''
4451       for i in string.utfcharacters(p:gsub('%d', '')) do
4452         ss = ss .. '%d?' .. i
4453       end
4454       ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
4455       ss = ss:gsub('%.%d%?$', '%%.')
4456       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')

```

```

4457     if n == 0 then
4458         tex.sprint(
4459             [[\string\csname\space bbl@info\endcsname{New pattern: }]]
4460             .. p .. [[]])
4461         pats = pats .. ' ' .. p
4462     else
4463         tex.sprint(
4464             [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
4465             .. p .. [[]])
4466     end
4467 end
4468 lang.patterns(lg, pats)
4469 end
4470 }
4471 \endgroup
4472 \ifx\newattribute\@undefined\else
4473   \newattribute\bbl@attr@locale
4474   \AddBabelHook{luatex}{beforeextras}{%
4475     \setattribute\bbl@attr@locale\localeid}
4476 \fi
4477 \def\BabelStringsDefault{unicode}
4478 \let\luabbl@stop\relax
4479 \AddBabelHook{luatex}{encodedcommands}{%
4480   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
4481   \ifx\bbl@tempa\bbl@tempb\else
4482     \directlua{Babel.begin_process_input()}%
4483     \def\luabbl@stop{%
4484       \directlua{Babel.end_process_input()}}%
4485   \fi}%
4486 \AddBabelHook{luatex}{stopcommands}{%
4487   \luabbl@stop
4488   \let\luabbl@stop\relax}
4489 \AddBabelHook{luatex}{patterns}{%
4490   \@ifundefined{bbl@hyphendata@the\language}%
4491   {\def\bbl@elt##1##2##3##4{%
4492     \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
4493     \def\bbl@tempb{##3}%
4494     \ifx\bbl@tempb\@empty\else % if not a synonymous
4495       \def\bbl@tempc{##3}{##4}%
4496     \fi
4497     \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4498   \fi}%
4499   \bbl@languages
4500   \@ifundefined{bbl@hyphendata@the\language}%
4501   {\bbl@info{No hyphenation patterns were set for\%
4502     language '#2'. Reported}}%
4503   {\expandafter\expandafter\expandafter\bbl@luapatterns
4504     \csname bbl@hyphendata@the\language\endcsname}}}%
4505   \@ifundefined{bbl@patterns@}{}%
4506   \begingroup
4507     \bbl@xin@{, \number\language,}{, \bbl@pttnlist}%
4508   \ifin@
4509     \ifx\bbl@patterns@\@empty\else
4510       \directlua{ Babel.addpatterns(
4511         [[\bbl@patterns@]], \number\language) }%
4512     \fi
4513     \@ifundefined{bbl@patterns@#1}%
4514     \@empty
4515     {\directlua{ Babel.addpatterns(

```

```

4516          [[\space\csname bbl@patterns@#1\endcsname]],
4517          \number\language) }}%
4518      \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
4519      \fi
4520  \endgroup}%
4521  \bbl@exp{%
4522      \bbl@ifunset{\bbl@prehc@\languagename}{}%
4523      {\bbl@ifblank{\bbl@cs{\prehc@\languagename}}{}}%
4524      {\prehyphenchar=\bbl@c1{\prehc}\relax}}}%

```

`\babelpatterns` This macro adds patterns. Two macros are used to store them: `\bbl@patterns@` for the global ones and `\bbl@patterns@<lang>` for language ones. We make sure there is a space between words when multiple commands are used.

```

4525 \@onlypreamble\babelpatterns
4526 \AtEndOfPackage{%
4527   \newcommand\babelpatterns[2][\@empty]{%
4528     \ifx\bbl@patterns@relax
4529       \let\bbl@patterns@\@empty
4530     \fi
4531     \ifx\bbl@pttnlist\@empty\else
4532       \bbl@warning{%
4533         You must not intermingle \string\selectlanguage\space and\%
4534         \string\babelpatterns\space or some patterns will not\%
4535         be taken into account. Reported}%
4536     \fi
4537     \ifx\@empty#1%
4538       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
4539     \else
4540       \edef\bbl@tempb{\zap@space#1 \@empty}%
4541       \bbl@for\bbl@tempa\bbl@tempb{%
4542         \bbl@fixname\bbl@tempa
4543         \bbl@iflanguage\bbl@tempa{%
4544           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
4545             \ifundefined{\bbl@patterns@\bbl@tempa}%
4546               \@empty
4547             {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
4548             #2}}}%
4549     \fi}}

```

## 13.4 Southeast Asian scripts

First, some general code for line breaking, used by `\babelposthyphenation`.

*In progress.* Replace regular (ie, implicit) discretionaries by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionaries are not touched.

For the moment, only 3 SA languages are activated by default (see Unicode UAX 14).

```

4550 \directlua{
4551   Babel = Babel or {}
4552   Babel.linebreaking = Babel.linebreaking or {}
4553   Babel.linebreaking.before = {}
4554   Babel.linebreaking.after = {}
4555   Babel.locale = {} % Free to use, indexed with \localeid
4556   function Babel.linebreaking.add_before(func)
4557     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
4558     table.insert(Babel.linebreaking.before , func)
4559   end
4560   function Babel.linebreaking.add_after(func)

```

```

4561 tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
4562 table.insert(Babel.linebreaking.after, func)
4563 end
4564 }
4565 \def\bbl@intraspace#1 #2 #3\@{%
4566 \directlua{
4567   Babel = Babel or {}
4568   Babel.intraspaces = Babel.intraspaces or {}
4569   Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
4570     {b = #1, p = #2, m = #3}
4571   Babel.locale_props[\the\localeid].intraspace = %
4572     {b = #1, p = #2, m = #3}
4573 }}
4574 \def\bbl@intrapenalty#1\@{%
4575 \directlua{
4576   Babel = Babel or {}
4577   Babel.intrapenalties = Babel.intrapenalties or {}
4578   Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
4579   Babel.locale_props[\the\localeid].intrapenalty = #1
4580 }}
4581 \begingroup
4582 \catcode`\%=12
4583 \catcode`\^=14
4584 \catcode`\'=12
4585 \catcode`\~=12
4586 \gdef\bbl@seaintraspace{^
4587 \let\bbl@seaintraspace\relax
4588 \directlua{
4589   Babel = Babel or {}
4590   Babel.sea_enabled = true
4591   Babel.sea_ranges = Babel.sea_ranges or {}
4592   function Babel.set_chranges (script, chrng)
4593     local c = 0
4594     for s, e in string.gmatch(chrng..' ', '(.)%.%.(-)%s') do
4595       Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
4596       c = c + 1
4597     end
4598   end
4599   function Babel.sea_disc_to_space (head)
4600     local sea_ranges = Babel.sea_ranges
4601     local last_char = nil
4602     local quad = 655360 ^^ 10 pt = 655360 = 10 * 65536
4603     for item in node.traverse(head) do
4604       local i = item.id
4605       if i == node.id'glyph' then
4606         last_char = item
4607       elseif i == 7 and item.subtype == 3 and last_char
4608         and last_char.char > 0x0C99 then
4609         quad = font.getfont(last_char.font).size
4610         for lg, rg in pairs(sea_ranges) do
4611           if last_char.char > rg[1] and last_char.char < rg[2] then
4612             lg = lg:sub(1, 4) ^^ Remove trailing number of, eg, Cyr11
4613             local intraspace = Babel.intraspaces[lg]
4614             local intrapenalty = Babel.intrapenalties[lg]
4615             local n
4616             if intrapenalty ~= 0 then
4617               n = node.new(14, 0) ^^ penalty
4618               n.penalty = intrapenalty
4619               node.insert_before(head, item, n)

```

```

4620         end
4621         n = node.new(12, 13)      ^^ (glue, spaceskip)
4622         node.setglue(n, intraspace.b * quad,
4623                     intraspace.p * quad,
4624                     intraspace.m * quad)
4625         node.insert_before(head, item, n)
4626         node.remove(head, item)
4627     end
4628 end
4629 end
4630 end
4631 end
4632 }^^
4633 \bbl@luahyphenate}
4634 \catcode`\%=14
4635 \gdef\bbl@cjkintraspac{%
4636   \let\bbl@cjkintraspac\relax
4637   \directlua{
4638     Babel = Babel or {}
4639     require'babel-data-cjk.lua'
4640     Babel.cjk_enabled = true
4641     function Babel.cjk_linebreak(head)
4642       local GLYPH = node.id'glyph'
4643       local last_char = nil
4644       local quad = 655360      % 10 pt = 655360 = 10 * 65536
4645       local last_class = nil
4646       local last_lang = nil
4647
4648       for item in node.traverse(head) do
4649         if item.id == GLYPH then
4650
4651           local lang = item.lang
4652
4653           local LOCALE = node.get_attribute(item,
4654             luatexbase.registernumber'bbl@attr@locale')
4655           local props = Babel.locale_props[LOCALE]
4656
4657           local class = Babel.cjk_class[item.char].c
4658
4659           if class == 'cp' then class = 'cl' end % ]) as CL
4660           if class == 'id' then class = 'I' end
4661
4662           local br = 0
4663           if class and last_class and Babel.cjk_breaks[last_class][class] then
4664             br = Babel.cjk_breaks[last_class][class]
4665           end
4666
4667           if br == 1 and props.linebreak == 'c' and
4668             lang ~= \the\l@nohyphenation\space and
4669             last_lang ~= \the\l@nohyphenation then
4670             local intrapenalty = props.intrapenalty
4671             if intrapenalty ~= 0 then
4672               local n = node.new(14, 0)      % penalty
4673               n.penalty = intrapenalty
4674               node.insert_before(head, item, n)
4675             end
4676             local intraspace = props.intraspace
4677             local n = node.new(12, 13)      % (glue, spaceskip)
4678             node.setglue(n, intraspace.b * quad,

```

```

4679             intraspace.p * quad,
4680             intraspace.m * quad)
4681         node.insert_before(head, item, n)
4682     end
4683
4684     quad = font.getfont(item.font).size
4685     last_class = class
4686     last_lang = lang
4687     else % if penalty, glue or anything else
4688         last_class = nil
4689     end
4690 end
4691 lang.hyphenate(head)
4692 end
4693 }%
4694 \bbl@luahyphenate}
4695 \gdef\bbl@luahyphenate{%
4696 \let\bbl@luahyphenate\relax
4697 \directlua{
4698     luatexbase.add_to_callback('hyphenate',
4699     function (head, tail)
4700         if Babel.linebreaking.before then
4701             for k, func in ipairs(Babel.linebreaking.before) do
4702                 func(head)
4703             end
4704         end
4705         if Babel.cjk_enabled then
4706             Babel.cjk_linebreak(head)
4707         end
4708         lang.hyphenate(head)
4709         if Babel.linebreaking.after then
4710             for k, func in ipairs(Babel.linebreaking.after) do
4711                 func(head)
4712             end
4713         end
4714         if Babel.sea_enabled then
4715             Babel.sea_disc_to_space(head)
4716         end
4717     end,
4718     'Babel.hyphenate')
4719 }
4720 }
4721 \endgroup
4722 \def\bbl@provide@intraspace{%
4723 \bbl@ifunset{\bbl@intsp@language}{}%
4724 {\expandafter\ifx\csname bbl@intsp@language\endcsname\@empty\else
4725 \bbl@xin@{\bbl@cl{lbrk}}{c}%
4726 \ifin@ % cjk
4727 \bbl@cjkintraspace
4728 \directlua{
4729     Babel = Babel or {}
4730     Babel.locale_props = Babel.locale_props or {}
4731     Babel.locale_props[\the\localeid].linebreak = 'c'
4732 }%
4733 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}{\@}%
4734 \ifx\bbl@KVP@intrapenalty\@nil
4735 \bbl@intrapenalty0\@@
4736 \fi
4737 \else % sea

```



```

4738      \bbl@seaintraspace
4739      \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\bbl@cl{intsp}\bbl@cl{intsp}}
4740      \directlua{
4741          Babel = Babel or {}
4742          Babel.sea_ranges = Babel.sea_ranges or {}
4743          Babel.set_chranges('\bbl@cl{sbcpr}',
4744                          '\bbl@cl{chrng}')
4745      }%
4746      \ifx\bbl@KVP@intrapenalty\@nil
4747          \bbl@intrapenalty0\@@
4748      \fi
4749      \fi
4750      \fi
4751      \ifx\bbl@KVP@intrapenalty\@nil\else
4752          \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4753      \fi}}

```

### 13.5 CJK line breaking

Minimal line breaking for CJK scripts, mainly intended for simple documents and short texts as a secondary language. Only line breaking, with a little stretching for justification, without any attempt to adjust the spacing. It is based on (but does not strictly follow) the Unicode algorithm.

We first need a little table with the corresponding line breaking properties. A few characters have an additional key for the width (fullwidth vs. halfwidth), not yet used. There is a separate file, defined below.

*Work in progress.*

Common stuff.

```

4754 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4755 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4756 \DisableBabelHook{babel-fontspec}
4757 <<Font selection>>

```

### 13.6 Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a short function which just traverse the node list to carry out the replacements. The table `loc_to_scr` gets the locale from a script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the `\language` and the `\localeid` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with `/` maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```

4758 \directlua{
4759 Babel.script_blocks = {
4760   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
4761              {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
4762   ['Armn'] = {{0x0530, 0x058F}},
4763   ['Beng'] = {{0x0980, 0x09FF}},
4764   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
4765   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
4766   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
4767              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
4768   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
4769   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
4770              {0xAB00, 0xAB2F}},

```

```

4771 ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
4772 % Don't follow strictly Unicode, which places some Coptic letters in
4773 % the 'Greek and Coptic' block
4774 ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
4775 ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
4776             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
4777             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
4778             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
4779             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
4780             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
4781 ['Hebr'] = {{0x0590, 0x05FF}},
4782 ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
4783             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
4784 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
4785 ['Knda'] = {{0x0C80, 0x0CFF}},
4786 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
4787             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
4788             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
4789 ['Laoo'] = {{0x0E80, 0x0EFF}},
4790 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
4791             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
4792             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
4793 ['Mahj'] = {{0x11150, 0x1117F}},
4794 ['Mlym'] = {{0x0D00, 0x0D7F}},
4795 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
4796 ['Orya'] = {{0x0B00, 0x0B7F}},
4797 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
4798 ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
4799 ['Taml'] = {{0x0B80, 0x0BFF}},
4800 ['Telu'] = {{0x0C00, 0x0C7F}},
4801 ['Tfng'] = {{0x2D30, 0x2D7F}},
4802 ['Thai'] = {{0x0E00, 0x0E7F}},
4803 ['Tibt'] = {{0x0F00, 0x0FFF}},
4804 ['Vaii'] = {{0xA500, 0xA63F}},
4805 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
4806 }
4807
4808 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
4809 Babel.script_blocks.Hant = Babel.script_blocks.Hans
4810 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
4811
4812 function Babel.locale_map(head)
4813   if not Babel.locale_mapped then return head end
4814
4815   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
4816   local GLYPH = node.id('glyph')
4817   local inmath = false
4818   local toloc_save
4819   for item in node.traverse(head) do
4820     local toloc
4821     if not inmath and item.id == GLYPH then
4822       % Optimization: build a table with the chars found
4823       if Babel.chr_to_loc[item.char] then
4824         toloc = Babel.chr_to_loc[item.char]
4825       else
4826         for lc, maps in pairs(Babel.loc_to_scr) do
4827           for _, rg in pairs(maps) do
4828             if item.char >= rg[1] and item.char <= rg[2] then
4829               Babel.chr_to_loc[item.char] = lc

```

```

4830         toloc = lc
4831         break
4832     end
4833 end
4834 end
4835 end
4836 % Now, take action, but treat composite chars in a different
4837 % fashion, because they 'inherit' the previous locale. Not yet
4838 % optimized.
4839 if not toloc and
4840     (item.char >= 0x0300 and item.char <= 0x036F) or
4841     (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
4842     (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
4843     toloc = toloc_save
4844 end
4845 if toloc and toloc > -1 then
4846     if Babel.locale_props[toloc].lg then
4847         item.lang = Babel.locale_props[toloc].lg
4848         node.set_attribute(item, LOCALE, toloc)
4849     end
4850     if Babel.locale_props[toloc]['/'..item.font] then
4851         item.font = Babel.locale_props[toloc]['/'..item.font]
4852     end
4853     toloc_save = toloc
4854 end
4855 elseif not inmath and item.id == 7 then
4856     item.replace = item.replace and Babel.locale_map(item.replace)
4857     item.pre      = item.pre and Babel.locale_map(item.pre)
4858     item.post     = item.post and Babel.locale_map(item.post)
4859 elseif item.id == node.id'math' then
4860     inmath = (item.subtype == 0)
4861 end
4862 end
4863 return head
4864 end
4865 }

```

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

```

4866 \newcommand\babelcharproperty[1]{%
4867   \count@=#1\relax
4868   \ifvmode
4869     \expandafter\bbl@chprop
4870   \else
4871     \bbl@error{\string\babelcharproperty\space can be used only in\\%
4872               vertical mode (preamble or between paragraphs)}%
4873     {See the manual for futher info}%
4874   \fi}
4875 \newcommand\bbl@chprop[3][\the\count@]{%
4876   \@tempcnta=#1\relax
4877   \bbl@ifunset{\bbl@chprop@#2}%
4878   {\bbl@error{No property named '#2'. Allowed values are\\%
4879             direction (bc), mirror (bmg), and linebreak (lb)}%
4880    {See the manual for futher info}}%
4881   }%
4882   \loop
4883     \bbl@cs{chprop@#2}{#3}%
4884   \ifnum\count@<\@tempcnta
4885     \advance\count@\@ne

```

```

4886 \repeat}
4887 \def\bbl@chprop@direction#1{%
4888 \directlua{
4889   Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
4890   Babel.characters[\the\count@]['d'] = '#1'
4891 }}
4892 \let\bbl@chprop@bc\bbl@chprop@direction
4893 \def\bbl@chprop@mirror#1{%
4894 \directlua{
4895   Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
4896   Babel.characters[\the\count@]['m'] = '\number#1'
4897 }}
4898 \let\bbl@chprop@bmg\bbl@chprop@mirror
4899 \def\bbl@chprop@linebreak#1{%
4900 \directlua{
4901   Babel.Babel.cjk_characters[\the\count@] = Babel.Babel.cjk_characters[\the\count@] or {}
4902   Babel.Babel.cjk_characters[\the\count@]['c'] = '#1'
4903 }}
4904 \let\bbl@chprop@lb\bbl@chprop@linebreak
4905 \def\bbl@chprop@locale#1{%
4906 \directlua{
4907   Babel.chr_to_loc = Babel.chr_to_loc or {}
4908   Babel.chr_to_loc[\the\count@] =
4909     \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
4910 }}

```

Post-handling hyphenation patterns for non-standard rules, like ff to ff-f. There are still some issues with speed (not very slow, but still slow).

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: `str_to_nodes` converts the string returned by a function to a node list, taking the node at base as a model (font, language, etc.); `fetch_word` fetches a series of glyphs and discretionaries, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck). `post_hyphenate_replace` is the callback applied after `lang.hyphenate`. This means the automatic hyphenation points are known. As empty captures return a byte position (as explained in the `luatex` manual), we must convert it to a utf8 position. With `first`, the last byte can be the leading byte in a utf8 sequence, so we just remove it and add 1 to the resulting length. With `last` we must take into account the capture position points to the next character. Here `word_head` points to the starting node of the text to be matched.

```

4911 \begingroup
4912 \catcode`\#=12
4913 \catcode`\%=12
4914 \catcode`\&=14
4915 \directlua{
4916   Babel.linebreaking.replacements = {}
4917
4918   function Babel.str_to_nodes(fn, matches, base)
4919     local n, head, last
4920     if fn == nil then return nil end
4921     for s in string.utfvalues(fn(matches)) do
4922       if base.id == 7 then
4923         base = base.replace
4924       end
4925       n = node.copy(base)
4926       n.char = s
4927       if not head then
4928         head = n
4929       else

```

```

4930         last.next = n
4931     end
4932     last = n
4933 end
4934 return head
4935 end
4936
4937 function Babel.fetch_word(head, funct)
4938     local word_string = ''
4939     local word_nodes = {}
4940     local lang
4941     local item = head
4942
4943     while item do
4944
4945         if item.id == 29
4946             and not(item.char == 124) && ie, not |
4947             and not(item.char == 61) && ie, not =
4948             and (item.lang == lang or lang == nil) then
4949             lang = lang or item.lang
4950             word_string = word_string .. unicode.utf8.char(item.char)
4951             word_nodes[#word_nodes+1] = item
4952
4953         elseif item.id == 7 and item.subtype == 2 then
4954             word_string = word_string .. '='
4955             word_nodes[#word_nodes+1] = item
4956
4957         elseif item.id == 7 and item.subtype == 3 then
4958             word_string = word_string .. '|'
4959             word_nodes[#word_nodes+1] = item
4960
4961         elseif word_string == '' then
4962             && pass
4963
4964         else
4965             return word_string, word_nodes, item, lang
4966         end
4967
4968         item = item.next
4969     end
4970 end
4971
4972 function Babel.post_hyphenate_replace(head)
4973     local u = unicode.utf8
4974     local lbkr = Babel.linebreaking.replacements
4975     local word_head = head
4976
4977     while true do
4978         local w, wn, nw, lang = Babel.fetch_word(word_head)
4979         if not lang then return head end
4980
4981         if not lbkr[lang] then
4982             break
4983         end
4984
4985         for k=1, #lbkr[lang] do
4986             local p = lbkr[lang][k].pattern
4987             local r = lbkr[lang][k].replace
4988

```

```

4989     while true do
4990         local matches = { u.match(w, p) }
4991         if #matches < 2 then break end
4992
4993         local first = table.remove(matches, 1)
4994         local last = table.remove(matches, #matches)
4995
4996         %% Fix offsets, from bytes to unicode.
4997         first = u.len(w:sub(1, first-1)) + 1
4998         last = u.len(w:sub(1, last-1))
4999
5000         local new %% used when inserting and removing nodes
5001         local changed = 0
5002
5003         %% This loop traverses the replace list and takes the
5004         %% corresponding actions
5005         for q = first, last do
5006             local crep = r[q-first+1]
5007             local char_node = wn[q]
5008             local char_base = char_node
5009
5010             if crep and crep.data then
5011                 char_base = wn[crep.data+first-1]
5012             end
5013
5014             if crep == {} then
5015                 break
5016             elseif crep == nil then
5017                 changed = changed + 1
5018                 node.remove(head, char_node)
5019             elseif crep and (crep.pre or crep.no or crep.post) then
5020                 changed = changed + 1
5021                 d = node.new(7, 0) %% (disc, discretionary)
5022                 d.pre = Babel.str_to_nodes(crep.pre, matches, char_base)
5023                 d.post = Babel.str_to_nodes(crep.post, matches, char_base)
5024                 d.replace = Babel.str_to_nodes(crep.no, matches, char_base)
5025                 d.attr = char_base.attr
5026                 if crep.pre == nil then %% TeXbook p96
5027                     d.penalty = crep.penalty or tex.hyphenpenalty
5028                 else
5029                     d.penalty = crep.penalty or tex.exhyphenpenalty
5030                 end
5031                 head, new = node.insert_before(head, char_node, d)
5032                 node.remove(head, char_node)
5033                 if q == 1 then
5034                     word_head = new
5035                 end
5036             elseif crep and crep.string then
5037                 changed = changed + 1
5038                 local str = crep.string(matches)
5039                 if str == '' then
5040                     if q == 1 then
5041                         word_head = char_node.next
5042                     end
5043                     head, new = node.remove(head, char_node)
5044                 elseif char_node.id == 29 and u.len(str) == 1 then
5045                     char_node.char = string.utfvalue(str)
5046                 else
5047                     local n

```

```

5048         for s in string.utfvalues(str) do
5049             if char_node.id == 7 then
5050                 log('Automatic hyphens cannot be replaced, just removed.')
5051             else
5052                 n = node.copy(char_base)
5053             end
5054             n.char = s
5055             if q == 1 then
5056                 head, new = node.insert_before(head, char_node, n)
5057                 word_head = new
5058             else
5059                 node.insert_before(head, char_node, n)
5060             end
5061         end
5062
5063         node.remove(head, char_node)
5064     end %% string length
5065     end %% if char and char.string
5066 end %% for char in match
5067 if changed > 20 then
5068     texio.write('Too many changes. Ignoring the rest.')
5069 elseif changed > 0 then
5070     w, wn, nw = Babel.fetch_word(word_head)
5071 end
5072
5073     end %% for match
5074 end %% for patterns
5075 word_head = nw
5076 end %% for words
5077 return head
5078 end
5079
5080 %% The following functions belong to the next macro
5081
5082 %% This table stores capture maps, numbered consecutively
5083 Babel.capture_maps = {}
5084
5085 function Babel.capture_func(key, cap)
5086     local ret = "[" .. cap:gsub('{{[0-9]}}', "]]..m[%1]..[(") .. "]"
5087     ret = ret:gsub('{{[0-9]}|([^\]|+)|(-)}', Babel.capture_func_map)
5088     ret = ret:gsub("%[%[%]%]%.%", '')
5089     ret = ret:gsub("%.%[%[%]%]%", '')
5090     return key .. "[=function(m) return ]] .. ret .. [[ end]]
5091 end
5092
5093 function Babel.capt_map(from, mapno)
5094     return Babel.capture_maps[mapno][from] or from
5095 end
5096
5097 %% Handle the {n|abc|ABC} syntax in captures
5098 function Babel.capture_func_map(capno, from, to)
5099     local froms = {}
5100     for s in string.utfcharacters(from) do
5101         table.insert(froms, s)
5102     end
5103     local cnt = 1
5104     table.insert(Babel.capture_maps, {})
5105     local mlen = table.getn(Babel.capture_maps)
5106     for s in string.utfcharacters(to) do

```

```

5107     Babel.capture_maps[mlen][froms[cnt]] = s
5108     cnt = cnt + 1
5109 end
5110 return "]]..Babel.capt_map(m[" .. capno .. "], " ..
5111     (mlen) .. ").. " .. "["
5112 end
5113
5114 }

```

Now the T<sub>E</sub>X high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the  $\{n\}$  syntax. For example, `pre={1}{1}`- becomes `function(m) return m[1]..m[1]..'-' end`, where `m` are the matches returned after applying the pattern. With a mapped capture the functions are similar to `function(m) return Babel.capt_map(m[1],1) end`, where the last argument identifies the mapping to be applied to `m[1]`. The way it is carried out is somewhat tricky, but the effect is not dissimilar to lua `load – save the code as string in a TeX macro, and expand this macro at the appropriate place. As \directlua does not take into account the current catcode of @, we just avoid this character in macro names (which explains the internal group, too).`

```

5115 \catcode`\#=6
5116 \gdef\babelposthyphenation#1#2#3{&%
5117     \bbl@activateposthyphen
5118     \begingroup
5119         \def\babeltempa{\bbl@add@list\babeltempb}&%
5120         \let\babeltempb\@empty
5121         \bbl@foreach{#3}{&%
5122             \bbl@ifsamestring{##1}{remove}&%
5123             {\bbl@add@list\babeltempb{nil}}&%
5124             {\directlua{
5125                 local rep = [[#1]]
5126                 rep = rep:gsub( '(no)%s*=%s*([^\s,]*)', Babel.capture_func)
5127                 rep = rep:gsub( '(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5128                 rep = rep:gsub( '(post)%s*=%s*([^\s,]*)', Babel.capture_func)
5129                 rep = rep:gsub( '(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5130                 tex.print([[\\string\babeltempa{}}] .. rep .. [[}}]])
5131             }}&%
5132         \directlua{
5133             local lbkr = Babel.linebreaking.replacements
5134             local u = unicode.utf8
5135             &% Convert pattern:
5136             local patt = string.gsub([[#2]], '%s', '')
5137             if not u.find(patt, '()', nil, true) then
5138                 patt = '()' .. patt .. '()'
5139             end
5140             patt = u.gsub(patt, '{(.)}',
5141                 function (n)
5142                     return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5143                 end)
5144             lbkr[\the\csname l@#1\endcsname] = lbkr[\the\csname l@#1\endcsname] or {}
5145             table.insert(lbkr[\the\csname l@#1\endcsname],
5146                 { pattern = patt, replace = { \babeltempb } })
5147         }&%
5148     \endgroup}
5149 \endgroup
5150 \def\bbl@activateposthyphen{%
5151     \let\bbl@activateposthyphen\relax
5152     \directlua{
5153         Babel.linebreaking.add_after(Babel.post_hyphenate_replace)

```



5154   }}

## 13.7 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.) with `bidi=basic`, without having to patch almost any macro where text direction is relevant.

`\@hangfrom` is useful in many contexts and it is redefined always with the `layout` option.

There are, however, a number of issues when the text direction is not the same as the box direction (as set by `\bodydir`), and when `\parbox` and `\hangindent` are involved.

Fortunately, latest releases of luatex simplify a lot the solution with `\shapemode`.

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, `tabular` seems to work (at least in simple cases) with `array`, `tabularx`, `hline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```

5155 \bbl@trace{Redefinitions for bidi layout}
5156 \ifx\@eqnnum\undefined\else
5157   \ifx\bbl@attr@dir\undefined\else
5158     \edef\@eqnnum{%
5159       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
5160       \unexpanded\expandafter{\@eqnnum}}%
5161   \fi
5162 \fi
5163 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
5164 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
5165   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
5166     \bbl@exp{%
5167       \mathdir\the\bodydir
5168       #1%           Once entered in math, set boxes to restore values
5169       \<ifmmode>%
5170       \everyvbox{%
5171         \the\everyvbox
5172         \bodydir\the\bodydir
5173         \mathdir\the\mathdir
5174         \everyhbox{\the\everyhbox}%
5175         \everyvbox{\the\everyvbox}}%
5176       \everyhbox{%
5177         \the\everyhbox
5178         \bodydir\the\bodydir
5179         \mathdir\the\mathdir
5180         \everyhbox{\the\everyhbox}%
5181         \everyvbox{\the\everyvbox}}%
5182       \<fi>}}%
5183   \def\@hangfrom#1{%
5184     \setbox\@tempboxa\hbox{{#1}}%
5185     \hangindent\wd\@tempboxa
5186     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5187       \shapemode\@ne
5188     \fi
5189     \noindent\box\@tempboxa}
5190 \fi
5191 \IfBabelLayout{tabular}
5192   {\let\bbl@OL@tabular\@tabular
5193    \bbl@replace\@tabular{$}\bbl@nextfake$}%
5194   \let\bbl@NL@tabular\@tabular

```

```

5195 \AtBeginDocument{%
5196   \ifx\bb1@NL@@tabular\@tabular\else
5197     \bb1@replace\@tabular{$}\bb1@nextfake$}%
5198   \let\bb1@NL@@tabular\@tabular
5199   \fi}}
5200 {}
5201 \IfBabelLayout{lists}
5202 {\let\bb1@OL@list\list
5203  \bb1@sreplace\list{\parshape}\bb1@listparshape}%
5204  \let\bb1@NL@list\list
5205  \def\bb1@listparshape#1#2#3{%
5206    \parshape #1 #2 #3 %
5207    \ifnum\bb1@getluadir{page}=\bb1@getluadir{par}\else
5208      \shapemode\tw@
5209    \fi}}
5210 {}
5211 \IfBabelLayout{graphics}
5212 {\let\bb1@pictresetdir\relax
5213  \def\bb1@pictsetdir{%
5214    \ifcase\bb1@thetextdir
5215      \let\bb1@pictresetdir\relax
5216    \else
5217      \textdir TLT\relax
5218      \def\bb1@pictresetdir{\textdir TRT\relax}%
5219    \fi}%
5220  \let\bb1@OL@@picture\@picture
5221  \let\bb1@OL@put\put
5222  \bb1@sreplace\@picture{\hskip-}\bb1@pictsetdir\hskip-}%
5223  \def\put(#1,#2)#3{% Not easy to patch. Better redefine.
5224    \@killglue
5225    \raise#2\unitlength
5226    \hb@xt@z@{\kern#1\unitlength\bb1@pictresetdir#3}\hss}}%
5227 \AtBeginDocument
5228 {\ifx\tikz@atbegin@node\@undefined\else
5229   \let\bb1@OL@pgfpicture\pgfpicture
5230   \bb1@sreplace\pgfpicture{\pgfpicturetrue}\bb1@pictsetdir\pgfpicturetrue}%
5231   \bb1@add\pgfsys@beginpicture{\bb1@pictsetdir}%
5232   \bb1@add\tikz@atbegin@node{\bb1@pictresetdir}%
5233   \fi}}
5234 {}

```

Implicitly reverses sectioning labels in bidi=basic-r, because the full stop is not in contact with L numbers any more. I think there must be a better way. Assumes bidi=basic, but there are some additional readjustments for bidi=default.

```

5235 \IfBabelLayout{counters}%
5236 {\let\bb1@OL@@textsuperscript\@textsuperscript
5237  \bb1@sreplace\@textsuperscript{\m@th}\m@th\mathdir\pagedir}%
5238  \let\bb1@latinarabic=\@arabic
5239  \let\bb1@OL@@arabic\@arabic
5240  \def\@arabic#1{\babelsublr{\bb1@latinarabic#1}}%
5241  \@ifpackagewith{babel}{\bidi=default}%
5242  {\let\bb1@asciroman=\@roman
5243   \let\bb1@OL@@roman\@roman
5244   \def\@roman#1{\babelsublr{\ensureascii{\bb1@asciroman#1}}}%
5245   \let\bb1@asciiRoman=\@Roman
5246   \let\bb1@OL@@roman\@Roman
5247   \def\@Roman#1{\babelsublr{\ensureascii{\bb1@asciiRoman#1}}}%
5248   \let\bb1@OL@labelenumii\labelenumii
5249   \def\labelenumii{}\theenumii}%

```

```

5250 \let\bbl@OL@p@enumiii\p@enumiii
5251 \def\p@enumiii{\p@enumii}\theenumii({}){}{}
5252 <<Footnote changes>>
5253 \IfBabelLayout{footnotes}%
5254 {\let\bbl@OL@footnote\footnote
5255 \BabelFootnote\footnote\language{}{}}%
5256 \BabelFootnote\localfootnote\language{}{}}%
5257 \BabelFootnote\mainfootnote{}{}{}
5258 {}

```

Some  $\LaTeX$  macros use internally the math mode for text formatting. They have very little in common and are grouped here, as a single option.

```

5259 \IfBabelLayout{extras}%
5260 {\let\bbl@OL@underline\underline
5261 \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
5262 \let\bbl@OL@LaTeX2e\LaTeX2e
5263 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
5264 \if b\expandafter\@car\@fseries\@nil\boldmath\fi
5265 \babelsublr{%
5266 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}
5267 {}
5268 </luatex>

```

### 13.8 Auto bidi with basic and basic-r

The file `babel-data-bidi.lua` currently only contains data. It is a large and boring file and it is not shown here (see the generated file), but here is a sample:

```

[0x25]={d='et'},
[0x26]={d='on'},
[0x27]={d='on'},
[0x28]={d='on', m=0x29},
[0x29]={d='on', m=0x28},
[0x2A]={d='on'},
[0x2B]={d='es'},
[0x2C]={d='cs'},

```

For the meaning of these codes, see the Unicode standard.

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

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

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, *what* they do and *why*, and not only *how*), but I think (or I hope) I've managed to understand them.

In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually *two* R modes (set explicitly in Unicode with RLM and ALM). In `babel` the `dir` is set by a higher protocol based on the language/script, which in turn sets the correct `dir` (<l>, <r> or <al>).

From UAX#9: “Where available, markup should be used instead of the explicit formatting characters”. So, this simple version just ignores formatting characters. Actually, most of that annex is devoted to how to handle them.

BD14-BD16 are not implemented. Unicode (and the W3C) are making a great effort to deal with some special problematic cases in “streamed” plain text. I don’t think this is the way to go – particular issues should be fixed by a high level interface taking into account the needs of the document. And here is where luatex excels, because everything related to bidi writing is under our control.

```

5269 (*basic-r)
5270 Babel = Babel or {}
5271
5272 Babel.bidi_enabled = true
5273
5274 require('babel-data-bidi.lua')
5275
5276 local characters = Babel.characters
5277 local ranges = Babel.ranges
5278
5279 local DIR = node.id("dir")
5280
5281 local function dir_mark(head, from, to, outer)
5282   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
5283   local d = node.new(DIR)
5284   d.dir = '+' .. dir
5285   node.insert_before(head, from, d)
5286   d = node.new(DIR)
5287   d.dir = '-' .. dir
5288   node.insert_after(head, to, d)
5289 end
5290
5291 function Babel.bidi(head, ispar)
5292   local first_n, last_n      -- first and last char with nums
5293   local last_es              -- an auxiliary 'last' used with nums
5294   local first_d, last_d      -- first and last char in L/R block
5295   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):

```

5296   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
5297   local strong_lr = (strong == 'l') and 'l' or 'r'
5298   local outer = strong
5299
5300   local new_dir = false
5301   local first_dir = false
5302   local inmath = false
5303
5304   local last_lr
5305
5306   local type_n = ''
5307
5308   for item in node.traverse(head) do
5309
5310     -- three cases: glyph, dir, otherwise
5311     if item.id == node.id'glyph'
5312       or (item.id == 7 and item.subtype == 2) then
5313
5314       local itemchar

```

```

5315     if item.id == 7 and item.subtype == 2 then
5316         itemchar = item.replace.char
5317     else
5318         itemchar = item.char
5319     end
5320     local chardata = characters[itemchar]
5321     dir = chardata and chardata.d or nil
5322     if not dir then
5323         for nn, et in ipairs(ranges) do
5324             if itemchar < et[1] then
5325                 break
5326             elseif itemchar <= et[2] then
5327                 dir = et[3]
5328                 break
5329             end
5330         end
5331     end
5332     dir = dir or 'l'
5333     if inmath then dir = ('TRT' == tex.mathdir) and 'r' or 'l' end

```

Next is based on the assumption babel sets the language AND switches the script with its dir. We treat a language block as a separate Unicode sequence. The following piece of code is executed at the first glyph after a ‘dir’ node. We don’t know the current language until then. This is not exactly true, as the math mode may insert explicit dirs in the node list, so, for the moment there is a hack by brute force (just above).

```

5334     if new_dir then
5335         attr_dir = 0
5336         for at in node.traverse(item.attr) do
5337             if at.number == luatexbase.registernumber'bbl@attr@dir' then
5338                 attr_dir = at.value % 3
5339             end
5340         end
5341         if attr_dir == 1 then
5342             strong = 'r'
5343         elseif attr_dir == 2 then
5344             strong = 'al'
5345         else
5346             strong = 'l'
5347         end
5348         strong_lr = (strong == 'l') and 'l' or 'r'
5349         outer = strong_lr
5350         new_dir = false
5351     end
5352
5353     if dir == 'nsm' then dir = strong end -- W1

```

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

```

5354     dir_real = dir -- We need dir_real to set strong below
5355     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:

```

5356     if strong == 'al' then
5357         if dir == 'en' then dir = 'an' end -- W2
5358         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
5359         strong_lr = 'r' -- W3
5360     end

```

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

```

5361 elseif item.id == node.id'dir' and not inmath then
5362     new_dir = true
5363     dir = nil
5364 elseif item.id == node.id'math' then
5365     inmath = (item.subtype == 0)
5366 else
5367     dir = nil          -- Not a char
5368 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>.

```

5369 if dir == 'en' or dir == 'an' or dir == 'et' then
5370     if dir ~= 'et' then
5371         type_n = dir
5372     end
5373     first_n = first_n or item
5374     last_n = last_es or item
5375     last_es = nil
5376 elseif dir == 'es' and last_n then -- W3+W6
5377     last_es = item
5378 elseif dir == 'cs' then          -- it's right - do nothing
5379     if first_n then -- & if dir = any but en, et, an, es, cs, inc nil
5380         if strong_lr == 'r' and type_n ~= '' then
5381             dir_mark(head, first_n, last_n, 'r')
5382         elseif strong_lr == 'l' and first_d and type_n == 'an' then
5383             dir_mark(head, first_n, last_n, 'r')
5384             dir_mark(head, first_d, last_d, outer)
5385             first_d, last_d = nil, nil
5386         elseif strong_lr == 'l' and type_n ~= '' then
5387             last_d = last_n
5388         end
5389         type_n = ''
5390         first_n, last_n = nil, nil
5391     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:

```

5392 if dir == 'l' or dir == 'r' then
5393     if dir ~= outer then
5394         first_d = first_d or item
5395         last_d = item
5396     elseif first_d and dir ~= strong_lr then
5397         dir_mark(head, first_d, last_d, outer)
5398         first_d, last_d = nil, nil
5399     end
5400 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.

```

5401   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
5402       item.char = characters[item.char] and
5403           characters[item.char].m or item.char
5404   elseif (dir or new_dir) and last_lr ~= item then
5405       local mir = outer .. strong_lr .. (dir or outer)
5406       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
5407           for ch in node.traverse(node.next(last_lr)) do
5408               if ch == item then break end
5409               if ch.id == node.id'glyph' and characters[ch.char] then
5410                   ch.char = characters[ch.char].m or ch.char
5411               end
5412           end
5413       end
5414   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).

```

5415   if dir == 'l' or dir == 'r' then
5416       last_lr = item
5417       strong = dir_real          -- Don't search back - best save now
5418       strong_lr = (strong == 'l') and 'l' or 'r'
5419   elseif new_dir then
5420       last_lr = nil
5421   end
5422 end

```

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

```

5423   if last_lr and outer == 'r' then
5424       for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
5425           if characters[ch.char] then
5426               ch.char = characters[ch.char].m or ch.char
5427           end
5428       end
5429   end
5430   if first_n then
5431       dir_mark(head, first_n, last_n, outer)
5432   end
5433   if first_d then
5434       dir_mark(head, first_d, last_d, outer)
5435   end

```

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

```

5436   return node.prev(head) or head
5437 end
5438 </basic-r>

```

And here the Lua code for bidi=basic:

```

5439 <(*basic)
5440 Babel = Babel or {}
5441
5442 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
5443
5444 Babel.fontmap = Babel.fontmap or {}
5445 Babel.fontmap[0] = {}          -- 1

```

```

5446 Babel.fontmap[1] = {}      -- r
5447 Babel.fontmap[2] = {}      -- al/an
5448
5449 Babel.bidi_enabled = true
5450 Babel.mirroring_enabled = true
5451
5452 require('babel-data-bidi.lua')
5453
5454 local characters = Babel.characters
5455 local ranges = Babel.ranges
5456
5457 local DIR = node.id('dir')
5458 local GLYPH = node.id('glyph')
5459
5460 local function insert_implicit(head, state, outer)
5461   local new_state = state
5462   if state.sim and state.eim and state.sim ~= state.eim then
5463     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
5464     local d = node.new(DIR)
5465     d.dir = '+' .. dir
5466     node.insert_before(head, state.sim, d)
5467     local d = node.new(DIR)
5468     d.dir = '-' .. dir
5469     node.insert_after(head, state.eim, d)
5470   end
5471   new_state.sim, new_state.eim = nil, nil
5472   return head, new_state
5473 end
5474
5475 local function insert_numeric(head, state)
5476   local new
5477   local new_state = state
5478   if state.san and state.ean and state.san ~= state.ean then
5479     local d = node.new(DIR)
5480     d.dir = '+TLT'
5481     _, new = node.insert_before(head, state.san, d)
5482     if state.san == state.sim then state.sim = new end
5483     local d = node.new(DIR)
5484     d.dir = '-TLT'
5485     _, new = node.insert_after(head, state.ean, d)
5486     if state.ean == state.eim then state.eim = new end
5487   end
5488   new_state.san, new_state.ean = nil, nil
5489   return head, new_state
5490 end
5491
5492 -- TODO - \hbox with an explicit dir can lead to wrong results
5493 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
5494 -- was s made to improve the situation, but the problem is the 3-dir
5495 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
5496 -- well.
5497
5498 function Babel.bidi(head, ispar, hdir)
5499   local d -- d is used mainly for computations in a loop
5500   local prev_d = ''
5501   local new_d = false
5502
5503   local nodes = {}
5504   local outer_first = nil

```



```

5505 local inmath = false
5506
5507 local glue_d = nil
5508 local glue_i = nil
5509
5510 local has_en = false
5511 local first_et = nil
5512
5513 local ATDIR = luatexbase.registernumber'bbl@attr@dir'
5514
5515 local save_outer
5516 local temp = node.get_attribute(head, ATDIR)
5517 if temp then
5518     temp = temp % 3
5519     save_outer = (temp == 0 and 'l') or
5520                 (temp == 1 and 'r') or
5521                 (temp == 2 and 'al')
5522 elseif ispar then -- Or error? Shouldn't happen
5523     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
5524 else -- Or error? Shouldn't happen
5525     save_outer = ('TRT' == hdir) and 'r' or 'l'
5526 end
5527 -- when the callback is called, we are just _after_ the box,
5528 -- and the textdir is that of the surrounding text
5529 -- if not ispar and hdir ~= tex.textdir then
5530 --     save_outer = ('TRT' == hdir) and 'r' or 'l'
5531 -- end
5532 local outer = save_outer
5533 local last = outer
5534 -- 'al' is only taken into account in the first, current loop
5535 if save_outer == 'al' then save_outer = 'r' end
5536
5537 local fontmap = Babel.fontmap
5538
5539 for item in node.traverse(head) do
5540
5541     -- In what follows, #node is the last (previous) node, because the
5542     -- current one is not added until we start processing the neutrals.
5543
5544     -- three cases: glyph, dir, otherwise
5545     if item.id == GLYPH
5546         or (item.id == 7 and item.subtype == 2) then
5547
5548         local d_font = nil
5549         local item_r
5550         if item.id == 7 and item.subtype == 2 then
5551             item_r = item.replace -- automatic discs have just 1 glyph
5552         else
5553             item_r = item
5554         end
5555         local chardata = characters[item_r.char]
5556         d = chardata and chardata.d or nil
5557         if not d or d == 'nsm' then
5558             for nn, et in ipairs(ranges) do
5559                 if item_r.char < et[1] then
5560                     break
5561                 elseif item_r.char <= et[2] then
5562                     if not d then d = et[3]
5563                     elseif d == 'nsm' then d_font = et[3]

```

```

5564         end
5565         break
5566     end
5567 end
5568 end
5569 d = d or 'l'
5570
5571 -- A short 'pause' in bidi for mapfont
5572 d_font = d_font or d
5573 d_font = (d_font == 'l' and 0) or
5574           (d_font == 'nsm' and 0) or
5575           (d_font == 'r' and 1) or
5576           (d_font == 'al' and 2) or
5577           (d_font == 'an' and 2) or nil
5578 if d_font and fontmap and fontmap[d_font][item_r.font] then
5579     item_r.font = fontmap[d_font][item_r.font]
5580 end
5581
5582 if new_d then
5583     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
5584     if inmath then
5585         attr_d = 0
5586     else
5587         attr_d = node.get_attribute(item, ATDIR)
5588         attr_d = attr_d % 3
5589     end
5590     if attr_d == 1 then
5591         outer_first = 'r'
5592         last = 'r'
5593     elseif attr_d == 2 then
5594         outer_first = 'r'
5595         last = 'al'
5596     else
5597         outer_first = 'l'
5598         last = 'l'
5599     end
5600     outer = last
5601     has_en = false
5602     first_et = nil
5603     new_d = false
5604 end
5605
5606 if glue_d then
5607     if (d == 'l' and 'l' or 'r') ~= glue_d then
5608         table.insert(nodes, {glue_i, 'on', nil})
5609     end
5610     glue_d = nil
5611     glue_i = nil
5612 end
5613
5614 elseif item.id == DIR then
5615     d = nil
5616     new_d = true
5617
5618 elseif item.id == node.id'glue' and item.subtype == 13 then
5619     glue_d = d
5620     glue_i = item
5621     d = nil
5622

```

```

5623     elseif item.id == node.id'math' then
5624         inmath = (item.subtype == 0)
5625
5626     else
5627         d = nil
5628     end
5629
5630     -- AL <= EN/ET/ES      -- W2 + W3 + W6
5631     if last == 'al' and d == 'en' then
5632         d = 'an'          -- W3
5633     elseif last == 'al' and (d == 'et' or d == 'es') then
5634         d = 'on'          -- W6
5635     end
5636
5637     -- EN + CS/ES + EN      -- W4
5638     if d == 'en' and #nodes >= 2 then
5639         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
5640             and nodes[#nodes-1][2] == 'en' then
5641             nodes[#nodes][2] = 'en'
5642         end
5643     end
5644
5645     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
5646     if d == 'an' and #nodes >= 2 then
5647         if (nodes[#nodes][2] == 'cs')
5648             and nodes[#nodes-1][2] == 'an' then
5649             nodes[#nodes][2] = 'an'
5650         end
5651     end
5652
5653     -- ET/EN                  -- W5 + W7->l / W6->on
5654     if d == 'et' then
5655         first_et = first_et or (#nodes + 1)
5656     elseif d == 'en' then
5657         has_en = true
5658         first_et = first_et or (#nodes + 1)
5659     elseif first_et then      -- d may be nil here !
5660         if has_en then
5661             if last == 'l' then
5662                 temp = 'l'    -- W7
5663             else
5664                 temp = 'en'   -- W5
5665             end
5666         else
5667             temp = 'on'       -- W6
5668         end
5669         for e = first_et, #nodes do
5670             if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
5671         end
5672         first_et = nil
5673         has_en = false
5674     end
5675
5676     if d then
5677         if d == 'al' then
5678             d = 'r'
5679             last = 'al'
5680         elseif d == 'l' or d == 'r' then
5681             last = d

```

```

5682     end
5683     prev_d = d
5684     table.insert(nodes, {item, d, outer_first})
5685 end
5686
5687     outer_first = nil
5688
5689 end
5690
5691 -- TODO -- repeated here in case EN/ET is the last node. Find a
5692 -- better way of doing things:
5693 if first_et then      -- dir may be nil here !
5694     if has_en then
5695         if last == 'l' then
5696             temp = 'l'      -- W7
5697         else
5698             temp = 'en'     -- W5
5699         end
5700     else
5701         temp = 'on'        -- W6
5702     end
5703     for e = first_et, #nodes do
5704         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
5705     end
5706 end
5707
5708 -- dummy node, to close things
5709 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
5710
5711 ----- NEUTRAL -----
5712
5713 outer = save_outer
5714 last = outer
5715
5716 local first_on = nil
5717
5718 for q = 1, #nodes do
5719     local item
5720
5721     local outer_first = nodes[q][3]
5722     outer = outer_first or outer
5723     last = outer_first or last
5724
5725     local d = nodes[q][2]
5726     if d == 'an' or d == 'en' then d = 'r' end
5727     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
5728
5729     if d == 'on' then
5730         first_on = first_on or q
5731     elseif first_on then
5732         if last == d then
5733             temp = d
5734         else
5735             temp = outer
5736         end
5737         for r = first_on, q - 1 do
5738             nodes[r][2] = temp
5739             item = nodes[r][1]      -- MIRRORING
5740             if Babel.mirroring_enabled and item.id == GLYPH

```

```

5741         and temp == 'r' and characters[item.char] then
5742             local font_mode = font.fonts[item.font].properties.mode
5743             if font_mode ~= 'harf' and font_mode ~= 'plug' then
5744                 item.char = characters[item.char].m or item.char
5745             end
5746         end
5747     end
5748     first_on = nil
5749 end
5750
5751 if d == 'r' or d == 'l' then last = d end
5752 end
5753
5754 ----- IMPLICIT, REORDER -----
5755
5756 outer = save_outer
5757 last = outer
5758
5759 local state = {}
5760 state.has_r = false
5761
5762 for q = 1, #nodes do
5763     local item = nodes[q][1]
5764
5765     outer = nodes[q][3] or outer
5766
5767     local d = nodes[q][2]
5768
5769     if d == 'nsm' then d = last end           -- W1
5770     if d == 'en' then d = 'an' end
5771     local isdir = (d == 'r' or d == 'l')
5772
5773     if outer == 'l' and d == 'an' then
5774         state.san = state.san or item
5775         state.ean = item
5776     elseif state.san then
5777         head, state = insert_numeric(head, state)
5778     end
5779
5780     if outer == 'l' then
5781         if d == 'an' or d == 'r' then        -- im -> implicit
5782             if d == 'r' then state.has_r = true end
5783             state.sim = state.sim or item
5784             state.eim = item
5785         elseif d == 'l' and state.sim and state.has_r then
5786             head, state = insert_implicit(head, state, outer)
5787         elseif d == 'l' then
5788             state.sim, state.eim, state.has_r = nil, nil, false
5789         end
5790     else
5791         if d == 'an' or d == 'l' then
5792             if nodes[q][3] then -- nil except after an explicit dir
5793                 state.sim = item -- so we move sim 'inside' the group
5794             else
5795                 state.sim = state.sim or item
5796             end
5797             state.eim = item
5798         elseif d == 'r' and state.sim then

```

```

5800         head, state = insert_implicit(head, state, outer)
5801     elseif d == 'r' then
5802         state.sim, state.eim = nil, nil
5803     end
5804 end
5805
5806 if isdir then
5807     last = d          -- Don't search back - best save now
5808 elseif d == 'on' and state.san then
5809     state.san = state.san or item
5810     state.ean = item
5811 end
5812
5813 end
5814
5815 return node.prev(head) or head
5816 end
5817 </basic>

```

## 14 Data for CJK

It is a boring file and it is not shown here (see the generated file), but here is a sample:

```

[0x0021]={c='ex'},
[0x0024]={c='pr'},
[0x0025]={c='po'},
[0x0028]={c='op'},
[0x0029]={c='cp'},
[0x002B]={c='pr'},

```

For the meaning of these codes, see the Unicode standard.

## 15 The ‘nil’ language

This ‘language’ does nothing, except setting the hyphenation patterns to nohyphenation. For this language currently no special definitions are needed or available.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```

5818 <nil>
5819 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language]
5820 \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.

```

5821 \ifx\l@nil\@undefined
5822   \newlanguage\l@nil
5823   \@namedef{bbl@hyphendata@the\l@nil}{}{}{}% Remove warning
5824   \let\bbl@elt\relax
5825   \edef\bbl@languages{% Add it to the list of languages
5826     \bbl@languages\bbl@elt{nil}{the\l@nil}{}{}
5827 \fi

```

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

```

5828 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

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

```
\captionnil
\datenil 5829 \let\captionnil\@empty
5830 \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.

```
5831 \ldf@finish{nil}
5832 </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  $\text{\TeX}$ , 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  $\text{\TeX}$  sees, we need to set some category codes just to be able to change the definition of `\input`.

```
5833 (*bplain | blplain)
5834 \catcode`\{=1 % left brace is begin-group character
5835 \catcode`\}=2 % right brace is end-group character
5836 \catcode`\#=6 % hash mark is macro parameter character
```

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

```
5837 \openin 0 hyphen.cfg
5838 \ifeof0
5839 \else
5840 \let\input
```

Then `\input` is defined to forget about its argument and load `hyphen.cfg` instead. Once that’s done the original meaning of `\input` can be restored and the definition of `\a` can be forgotten.

```
5841 \def\input #1 {%
5842 \let\input\input
5843 \a hyphen.cfg
5844 \let\input\undefined
5845 }
5846 \fi
5847 </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`.

```
5848 \bplain\la plain.tex
5849 \bplain\la 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.

```
5850 \bplain\def\fmtname{babel-plain}
5851 \bplain\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`.

```
5852 \langle *Emulate LaTeX \rangle \equiv
5853 % == Code for plain ==
5854 \def\@empty{}
5855 \def\loadlocalcfg#1{%
5856   \openin0#1.cfg
5857   \ifeof0
5858     \closein0
5859   \else
5860     \closein0
5861     {\immediate\write16{*****}%
5862      \immediate\write16{* Local config file #1.cfg used}%
5863      \immediate\write16{*}%
5864     }
5865     \input #1.cfg\relax
5866   \fi
5867   \@endofldef}
```

## 16.3 General tools

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

```
5868 \long\def\@firstofone#1{#1}
5869 \long\def\@firstoftwo#1#2{#1}
5870 \long\def\@secondoftwo#1#2{#2}
5871 \def\@nnil{\@nil}
5872 \def\@gobbletwo#1#2{}
5873 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
5874 \def\@star@or@long#1{%
5875   \@ifstar
5876   {\let\l@ngrel@x\relax#1}%
5877   {\let\l@ngrel@x\long#1}}
5878 \let\l@ngrel@x\relax
5879 \def\@car#1#2\@nil{#1}
5880 \def\@cdr#1#2\@nil{#2}
5881 \let\@typeset@protect\relax
5882 \let\protected@edef\edef
5883 \long\def\@gobble#1{}
5884 \edef\@backslashchar{\expandafter\@gobble\string\}
5885 \def\strip@prefix#1>{}
5886 \def\g@addto@macro#1#2{{%
5887   \toks@\expandafter{#1#2}%
5888   \xdef#1{\the\toks@}}}
```



```

5889 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
5890 \def\@nameuse#1{\csname #1\endcsname}
5891 \def\@ifundefined#1{%
5892   \expandafter\ifx\csname#1\endcsname\relax
5893     \expandafter\@firstoftwo
5894   \else
5895     \expandafter\@secondoftwo
5896   \fi}
5897 \def\@expandtwoargs#1#2#3{%
5898   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
5899 \def\zap@space#1 #2{%
5900   #1%
5901   \ifx#2\@empty\else\expandafter\zap@space\fi
5902   #2}
5903 \let\bbl@trace\@gobble

```

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

```

5904 \ifx\@preamblecmds\@undefined
5905   \def\@preamblecmds{}
5906 \fi
5907 \def\@onlypreamble#1{%
5908   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
5909     \@preamblecmds\do#1}}
5910 \@onlypreamble\@onlypreamble

```

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

```

5911 \def\begin{document}{%
5912   \@begin{document}hook
5913   \global\let\@begin{document}hook\@undefined
5914   \def\do##1{\global\let##1\@undefined}%
5915   \@preamblecmds
5916   \global\let\do\noexpand}
5917 \ifx\@begin{document}hook\@undefined
5918   \def\@begin{document}hook{}
5919 \fi
5920 \@onlypreamble\@begin{document}hook
5921 \def\AtBeginDocument{\g@addto@macro\@begin{document}hook}

```

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

```

5922 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
5923 \@onlypreamble\AtEndOfPackage
5924 \def\@endofldf{}
5925 \@onlypreamble\@endofldf
5926 \let\bbl@afterlang\@empty
5927 \chardef\bbl@opt@hyphenmap\z@

```

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

```

5928 \catcode`\&=\z@
5929 \ifx&\if@files\@undefined
5930   \expandafter\let\csname if@files\endcsname
5931   \csname iffalse\endcsname
5932 \fi
5933 \catcode`\&=4

```

Mimick L<sup>A</sup>T<sub>E</sub>X's commands to define control sequences.

```

5934 \def\newcommand{\@star@or@long\new@command}
5935 \def\new@command#1{%
5936   \@testopt{\@newcommand#1}0}
5937 \def\@newcommand#1[#2]{%
5938   \@ifnextchar [{\@xargdef#1[#2]]%
5939     {\@argdef#1[#2]}}
5940 \long\def\@argdef#1[#2]#3{%
5941   \@yargdef#1\@ne{#2}{#3}}
5942 \long\def\@xargdef#1[#2][#3]#4{%
5943   \expandafter\def\expandafter#1\expandafter{%
5944     \expandafter\@protected@testopt\expandafter #1%
5945     \csname\string#1\expandafter\endcsname{#3}}%
5946   \expandafter\@yargdef \csname\string#1\endcsname
5947   \tw@{#2}{#4}}
5948 \long\def\@yargdef#1#2#3{%
5949   \@tempcnta#3\relax
5950   \advance \@tempcnta \@ne
5951   \let\@hash@\relax
5952   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
5953   \@tempcntb #2%
5954   \@whilenum\@tempcntb <\@tempcnta
5955   \do{%
5956     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
5957     \advance\@tempcntb \@ne}%
5958   \let\@hash@###
5959   \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
5960 \def\providecommand{\@star@or@long\provide@command}
5961 \def\provide@command#1{%
5962   \begingroup
5963     \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
5964   \endgroup
5965   \expandafter\ifundefined\@gtempa
5966     {\def\reserved@a{\new@command#1}}%
5967     {\let\reserved@a\relax
5968     \def\reserved@a{\new@command\reserved@a}}%
5969   \reserved@a}%
5970 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
5971 \def\declare@robustcommand#1{%
5972   \edef\reserved@a{\string#1}%
5973   \def\reserved@b{#1}%
5974   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
5975   \edef#1{%
5976     \ifx\reserved@a\reserved@b
5977       \noexpand\x@protect
5978       \noexpand#1%
5979     \fi
5980     \noexpand\protect
5981     \expandafter\noexpand\csname
5982       \expandafter\@gobble\string#1 \endcsname
5983   }%
5984   \expandafter\new@command\csname
5985     \expandafter\@gobble\string#1 \endcsname
5986 }
5987 \def\x@protect#1{%
5988   \ifx\protect\@typeset@protect\else
5989     \x@protect#1%
5990   \fi

```

```

5991 }
5992 \catcode`\&=\z@ % Trick to hide conditionals
5993 \def\@x@protect#1&fi#2#3{&fi\protect#1}

```

The following little macro `\in@` is taken from `latex.ltx`; it checks whether its first argument is part of its second argument. It uses the boolean `\in@`; allocating a new boolean inside conditionally executed code is not possible, hence the construct with the temporary definition of `\bbl@tempa`.

```

5994 \def\bbl@tempa{\csname newif\endcsname&fin@}
5995 \catcode`\&=4
5996 \ifx\in@\@undefined
5997 \def\in@#1#2{%
5998   \def\in@@##1#1##2##3\in@{%
5999     \ifx\in@@##2\in@false\else\in@true\fi}%
6000   \in@@#2#1\in@\in@@}
6001 \else
6002 \let\bbl@tempa\@empty
6003 \fi
6004 \bbl@tempa

```

$\text{\LaTeX}$  has a macro to check whether a certain package was loaded with specific options. The command has two extra arguments which are code to be executed in either the true or false case. This is used to detect whether the document needs one of the accents to be activated (activegrave and activeacute). For plain  $\text{\TeX}$  we assume that the user wants them to be active by default. Therefore the only thing we do is execute the third argument (the code for the true case).

```

6005 \def\@ifpackagewith#1#2#3#4{#3}

```

The  $\text{\LaTeX}$  macro `\@ifl@aded` checks whether a file was loaded. This functionality is not needed for plain  $\text{\TeX}$  but we need the macro to be defined as a no-op.

```

6006 \def\@ifl@aded#1#2#3#4{}

```

For the following code we need to make sure that the commands `\newcommand` and `\providecommand` exist with some sensible definition. They are not fully equivalent to their  $\text{\LaTeX} 2_{\epsilon}$  versions; just enough to make things work in plain  $\text{\TeX}$  environments.

```

6007 \ifx\@tempcnta\@undefined
6008 \csname newcount\endcsname\@tempcnta\relax
6009 \fi
6010 \ifx\@tempcntb\@undefined
6011 \csname newcount\endcsname\@tempcntb\relax
6012 \fi

```

To prevent wasting two counters in  $\text{\LaTeX} 2.09$  (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```

6013 \ifx\bye\@undefined
6014 \advance\count10 by -2\relax
6015 \fi
6016 \ifx\@ifnextchar\@undefined
6017 \def\@ifnextchar#1#2#3{%
6018   \let\reserved@d=#1%
6019   \def\reserved@a{#2}\def\reserved@b{#3}%
6020   \futurelet\@let@token\@ifnch}
6021 \def\@ifnch{%
6022   \ifx\@let@token\@sptoken
6023     \let\reserved@c\@xifnch
6024   \else
6025     \ifx\@let@token\reserved@d
6026       \let\reserved@c\reserved@a

```

```

6027     \else
6028     \let\reserved@c\reserved@b
6029     \fi
6030 \fi
6031 \reserved@c}
6032 \def\:\let\sptoken= } \: % this makes \@sptoken a space token
6033 \def\:\@xifnch} \expandafter\def\:\{\futurelet\@let@token\@ifnch}
6034 \fi
6035 \def\@testopt#1#2{%
6036   \ifnextchar[{\#1}{\#1[#2]}}
6037 \def\@protected@testopt#1{%
6038   \ifx\protect\@typeset@protect
6039     \expandafter\@testopt
6040   \else
6041     \@x@protect#1%
6042   \fi}
6043 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
6044   #2\relax}\fi}
6045 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
6046   \else\expandafter\@gobble\fi{#1}}

```

## 16.4 Encoding related macros

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

```

6047 \def\DeclareTextCommand{%
6048   \@dec@text@cmd\providecommand
6049 }
6050 \def\ProvideTextCommand{%
6051   \@dec@text@cmd\providecommand
6052 }
6053 \def\DeclareTextSymbol#1#2#3{%
6054   \@dec@text@cmd\chardef#1{#2}#3\relax
6055 }
6056 \def\@dec@text@cmd#1#2#3{%
6057   \expandafter\def\expandafter#2%
6058     \expandafter{%
6059       \csname#3-cmd\expandafter\endcsname
6060       \expandafter#2%
6061       \csname#3\string#2\endcsname
6062     }%
6063 %   \let\@ifdefinable\rc@ifdefinable
6064   \expandafter#1\csname#3\string#2\endcsname
6065 }
6066 \def\@current@cmd#1{%
6067   \ifx\protect\@typeset@protect\else
6068     \noexpand#1\expandafter\@gobble
6069   \fi
6070 }
6071 \def\@changed@cmd#1#2{%
6072   \ifx\protect\@typeset@protect
6073     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
6074       \expandafter\ifx\csname ?\string#1\endcsname\relax
6075         \expandafter\def\csname ?\string#1\endcsname{%
6076           \@changed@x@err{#1}%
6077         }%
6078       \fi
6079     \global\expandafter\let
6080     \csname\cf@encoding \string#1\expandafter\endcsname

```

```

6081         \csname ?\string#1\endcsname
6082     \fi
6083     \csname\cf@encoding\string#1%
6084         \expandafter\endcsname
6085 \else
6086     \noexpand#1%
6087 \fi
6088 }
6089 \def\@changed@x@err#1{%
6090     \errhelp{Your command will be ignored, type <return> to proceed}%
6091     \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
6092 \def\DeclareTextCommandDefault#1{%
6093     \DeclareTextCommand#1?%
6094 }
6095 \def\ProvideTextCommandDefault#1{%
6096     \ProvideTextCommand#1?%
6097 }
6098 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
6099 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
6100 \def\DeclareTextAccent#1#2#3{%
6101     \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
6102 }
6103 \def\DeclareTextCompositeCommand#1#2#3#4{%
6104     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
6105     \edef\reserved@b{\string##1}%
6106     \edef\reserved@c{%
6107         \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
6108     \ifx\reserved@b\reserved@c
6109         \expandafter\expandafter\expandafter\ifx
6110             \expandafter\@car\reserved@a\relax\relax\@nil
6111             \@text@composite
6112     \else
6113         \edef\reserved@b##1{%
6114             \def\expandafter\noexpand
6115                 \csname#2\string#1\endcsname####1{%
6116                 \noexpand\@text@composite
6117                 \expandafter\noexpand\csname#2\string#1\endcsname
6118                 ####1\noexpand\@empty\noexpand\@text@composite
6119                 {##1}%
6120             }%
6121         }%
6122         \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
6123     \fi
6124     \expandafter\def\csname\expandafter\string\csname
6125         #2\endcsname\string#1-\string#3\endcsname{#4}
6126 \else
6127     \errhelp{Your command will be ignored, type <return> to proceed}%
6128     \errmessage{\string\DeclareTextCompositeCommand\space used on
6129         inappropriate command \protect#1}
6130 \fi
6131 }
6132 \def\@text@composite#1#2#3\@text@composite{%
6133     \expandafter\@text@composite@x
6134     \csname\string#1-\string#2\endcsname
6135 }
6136 \def\@text@composite@x#1#2{%
6137     \ifx#1\relax
6138         #2%
6139     \else

```

```

6140      #1%
6141      \fi
6142 }
6143 %
6144 \def\@strip@args#1:#2-#3\@strip@args{#2}
6145 \def\DeclareTextComposite#1#2#3#4{%
6146   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
6147   \bgroup
6148     \lccode`\@=#4%
6149     \lowercase{%
6150   \egroup
6151   \reserved@a @%
6152 }%
6153 }
6154 %
6155 \def\UseTextSymbol#1#2{%
6156 %   \let\@curr@enc\cf@encoding
6157 %   \@use@text@encoding{#1}%
6158   #2%
6159 %   \@use@text@encoding\@curr@enc
6160 }
6161 \def\UseTextAccent#1#2#3{%
6162 %   \let\@curr@enc\cf@encoding
6163 %   \@use@text@encoding{#1}%
6164 %   #2{\@use@text@encoding\@curr@enc\selectfont#3}%
6165 %   \@use@text@encoding\@curr@enc
6166 }
6167 \def\@use@text@encoding#1{%
6168 %   \edef\font@encoding{#1}%
6169 %   \xdef\font@name{%
6170 %     \csname\curr@fontshape/\font@size\endcsname
6171 %   }%
6172 %   \pickup@font
6173 %   \font@name
6174 %   \@@enc@update
6175 }
6176 \def\DeclareTextSymbolDefault#1#2{%
6177   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
6178 }
6179 \def\DeclareTextAccentDefault#1#2{%
6180   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
6181 }
6182 \def\cf@encoding{OT1}

```

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

```

6183 \DeclareTextAccent{"}{OT1}{127}
6184 \DeclareTextAccent{'}{OT1}{19}
6185 \DeclareTextAccent{^}{OT1}{94}
6186 \DeclareTextAccent`}{OT1}{18}
6187 \DeclareTextAccent{~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for `PLAIN TEX`.

```

6188 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
6189 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
6190 \DeclareTextSymbol{\textquoteleft}{OT1}{`\'}
6191 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
6192 \DeclareTextSymbol{\i}{OT1}{16}
6193 \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`.

```
6194 \ifx\scriptsize\@undefined
6195   \let\scriptsize\sevenrm
6196 \fi
6197 % End of code for plain
6198 <</Emulate LaTeX>>
```

A proxy file:

```
6199 <plain>
6200 \input babel.def
6201 </plain>
```

## 17 Acknowledgements

I would like to thank all who volunteered as  $\beta$ -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

## References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national  $\LaTeX$  styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The  $\TeX$ book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport,  *$\LaTeX$ , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in:  $\TeX$ hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Hubert Partl, *German  $\TeX$* , *TUGboat* 9 (1988) #1, p. 70–72.
- [10] Joachim Schrod, *International  $\LaTeX$  is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [11] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using  $\LaTeX$* , Springer, 2002, p. 301–373.
- [12] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).