

# Babel

Version 3.41.1952  
2020/03/20

*Original author*  
Johannes L. Braams

*Current maintainer*  
Javier Bezos

Localization and  
internationalization

TeX

pdfTeX

LuaTeX

LuaHBTeX

XeTeX

# 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 . . . . .	8
1.7	Basic language selectors . . . . .	9
1.8	Auxiliary language selectors . . . . .	9
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	Selecting scripts . . . . .	35
1.21	Selecting directions . . . . .	36
1.22	Language attributes . . . . .	40
1.23	Hooks . . . . .	40
1.24	Languages supported by babel with ldf files . . . . .	41
1.25	Unicode character properties in luatex . . . . .	43
1.26	Tweaking some features . . . . .	43
1.27	Tips, workarounds, known issues and notes . . . . .	43
1.28	Current and future work . . . . .	45
1.29	Tentative and experimental code . . . . .	45
<b>2</b>	<b>Loading languages with language.dat</b>	<b>45</b>
2.1	Format . . . . .	46
<b>3</b>	<b>The interface between the core of babel and the language definition files</b>	<b>46</b>
3.1	Guidelines for contributed languages . . . . .	48
3.2	Basic macros . . . . .	48
3.3	Skeleton . . . . .	49
3.4	Support for active characters . . . . .	50
3.5	Support for saving macro definitions . . . . .	51
3.6	Support for extending macros . . . . .	51
3.7	Macros common to a number of languages . . . . .	51
3.8	Encoding-dependent strings . . . . .	52
<b>4</b>	<b>Changes</b>	<b>55</b>
4.1	Changes in babel version 3.9 . . . . .	55
<b>II</b>	<b>Source code</b>	<b>56</b>
<b>5</b>	<b>Identification and loading of required files</b>	<b>56</b>

<b>6</b>	<b>locale directory</b>	<b>56</b>
<b>7</b>	<b>Tools</b>	<b>57</b>
7.1	Multiple languages . . . . .	61
<b>8</b>	<b>The Package File (<math>\LaTeX</math>, babel.sty)</b>	<b>62</b>
8.1	base . . . . .	62
8.2	key=value options and other general option . . . . .	64
8.3	Conditional loading of shorthands . . . . .	66
8.4	Language options . . . . .	67
<b>9</b>	<b>The kernel of Babel (babel.def, common)</b>	<b>70</b>
9.1	Tools . . . . .	70
9.2	Hooks . . . . .	72
9.3	Setting up language files . . . . .	74
9.4	Shorthands . . . . .	76
9.5	Language attributes . . . . .	86
9.6	Support for saving macro definitions . . . . .	88
9.7	Short tags . . . . .	89
9.8	Hyphens . . . . .	89
9.9	Multiencoding strings . . . . .	91
9.10	Macros common to a number of languages . . . . .	97
9.11	Making glyphs available . . . . .	97
9.11.1	Quotation marks . . . . .	97
9.11.2	Letters . . . . .	98
9.11.3	Shorthands for quotation marks . . . . .	99
9.11.4	Umlauts and tremas . . . . .	100
9.12	Layout . . . . .	101
9.13	Load engine specific macros . . . . .	102
9.14	Creating languages . . . . .	102
<b>10</b>	<b>Adjusting the Babel behavior</b>	<b>116</b>
<b>11</b>	<b>The kernel of Babel (babel.def for <math>\LaTeX</math>only)</b>	<b>117</b>
11.1	The redefinition of the style commands . . . . .	117
11.2	Cross referencing macros . . . . .	118
11.3	Marks . . . . .	121
11.4	Preventing clashes with other packages . . . . .	122
11.4.1	ifthen . . . . .	122
11.4.2	varioref . . . . .	123
11.4.3	hhline . . . . .	123
11.4.4	hyperref . . . . .	124
11.4.5	fancyhdr . . . . .	124
11.5	Encoding and fonts . . . . .	125
11.6	Basic bidi support . . . . .	126
11.7	Local Language Configuration . . . . .	130
<b>12</b>	<b>Multiple languages (switch.def)</b>	<b>130</b>
12.1	Selecting the language . . . . .	131
12.2	Errors . . . . .	140
<b>13</b>	<b>Loading hyphenation patterns</b>	<b>142</b>
<b>14</b>	<b>Font handling with fontspec</b>	<b>146</b>

<b>15</b>	<b>Hooks for XeTeX and LuaTeX</b>	<b>151</b>
15.1	XeTeX . . . . .	151
15.2	Layout . . . . .	154
15.3	LuaTeX . . . . .	155
15.4	Southeast Asian scripts . . . . .	161
15.5	CJK line breaking . . . . .	165
15.6	Automatic fonts and ids switching . . . . .	165
15.7	Layout . . . . .	172
15.8	Auto bidi with basic and basic-r . . . . .	175
<b>16</b>	<b>Data for CJK</b>	<b>186</b>
<b>17</b>	<b>The ‘nil’ language</b>	<b>186</b>
<b>18</b>	<b>Support for Plain T<sub>E</sub>X (plain.def)</b>	<b>187</b>
18.1	Not renaming hyphen.tex . . . . .	187
18.2	Emulating some L <sub>A</sub> T <sub>E</sub> X features . . . . .	188
18.3	General tools . . . . .	188
18.4	Encoding related macros . . . . .	192
<b>19</b>	<b>Acknowledgements</b>	<b>195</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).
- 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, TeXLive, 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  $\LaTeX$ , the preamble of the document:

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

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

You can also set the main language explicitly, but it is discouraged except if there 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  $\LaTeX$   $\geq$  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 \babelfont, if used.)

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

**EXAMPLE** A trivial document is:

LUATEX/XETEX

```
\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.

## 1.6 Plain

In Plain, load languages styles with `\input` and then use `\begindocument` (the latter is defined by babel):

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

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

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

```
\input estonian.sty
\begindocument
```

**WARNING** Not all languages provide a sty file and some of them are not compatible with Plain.<sup>4</sup>

## 1.7 Basic language selectors

This section describes the commands to be used in the document to switch the language in multilingual documents. In most cases, only the two basic macros `\selectlanguage` and `\foreignlanguage` are necessary. The environments `otherlanguage`, `otherlanguage*` and `hyphenrules` are auxiliary, and described in the next section.

The main language is selected automatically when the document environment begins.

`\selectlanguage`  $\{\langle language \rangle\}$

When a user wants to switch from one language to another he can do so using the macro `\selectlanguage`. This macro takes the language, defined previously by a language definition file, as its argument. It calls several macros that should be defined in the language definition files to activate the special definitions for the language chosen:

```
\selectlanguage{german}
```

This command can be used as environment, too.

**NOTE** For “historical reasons”, a macro name is converted to a language name without the leading `\`; in other words, `\selectlanguage{\german}` is equivalent to `\selectlanguage{german}`. Using a macro instead of a “real” name is deprecated.

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

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

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

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

The command `\foreignlanguage` takes two arguments; the second argument is a phrase to be typeset according to the rules of the language named in its first one. This command (1) only switches the extra definitions and the hyphenation rules for the language, *not* the names and dates, (2) does not send information about the language to auxiliary files (i.e., the surrounding language is still in force), and (3) it works even if the language has not been set as package option (but in such a case it only sets the hyphenation patterns and a warning is shown). With the `bidirules` option, it also enters in horizontal mode (this is not done always for backwards compatibility).

## 1.8 Auxiliary language selectors

`\begin{otherlanguage}` { $\langle language \rangle$ } ... `\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*}` { $\langle language \rangle$ } ... `\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}` { $\langle language \rangle$ } ... `\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` { $\langle tag1 \rangle = \langle language1 \rangle, \langle tag2 \rangle = \langle language2 \rangle, \dots$ }

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

It defines `\text{<tag1>}{<text>}` to be `\foreignlanguage{<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

---

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

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

```
\babelensure{polish}
```

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

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

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

---

<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 are repeated at the beginning of the next line. You could then set:

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

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

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

`\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 german 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=**  $\langle char \rangle \langle char \rangle \dots$  | 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 3.9l</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 3.9l</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   main   select   other   other*

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

**New 3.9g** Sets the behavior of case mapping for hyphenation, provided the language defines it.<sup>10</sup> It can take the following values:

**off** deactivates this feature and no case mapping is applied;  
**first** sets it at the first switching commands in the current or parent scope (typically, when the aux file is first read and at `\begin{document}`), but also the first `\selectlanguage` in the preamble), and it's the default if a single language option has been stated;<sup>11</sup>  
**select** sets it only at `\selectlanguage`;  
**other** also sets it at `otherlanguage`;  
**other\*** also sets it at `otherlanguage*` as well as in heads and foots (if the option `headfoot` is used) and in auxiliary files (ie, at `\select@language`), and it's the default if several language options have been stated. The option `first` can be regarded as an optimized version of `other*` for monolingual documents.<sup>12</sup>

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

**New 3.14** Selects the bidi algorithm to be used in `luatex` and `xetex`. See sec. 1.21.

**layout=**

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

## 1.12 The base option

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

**\AfterBabelLanguage** `{⟨option-name⟩}{⟨code⟩}`

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

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

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

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

---

<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}

```

### 1.13 ini files

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

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

Most of them set the date, and many also the captions (Unicode and LICR). They will be evolving with the time to add more features (something to keep in mind if backward compatibility is important). The following section shows how to make use of them 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 follows:

**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 seems better, but still problematic).

**Devanagari** In luatex 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 luatex. On the other hand, xetex is better. The upcoming luatex will be based on luahtex, so Indic scripts will be rendered correctly with the option `Renderer=Harfbuzz` in `FONTSPEC`.

**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. The comment about Indic scripts and luatex also applies here. Some quick patterns could help, with something similar to:

```
\babelprovide[import,hyphenrules=+]{lao}
\babelpatterns[lao]{lᦺ lᦴ lᦵ lᦶ lᦷ lᦸ lᦹ} % Random
```

**East Asia scripts** Settings for either Simplified or Traditional should work out of the box. luatex does basic line breaking, but currently xetex does not (you may load `zhspacing`). 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>	bem	Bemba
agq	Aghem	bez	Bena
ak	Akan	bg	Bulgarian <sup>ul</sup>
am	Amharic <sup>ul</sup>	bm	Bambara
ar	Arabic <sup>ul</sup>	bn	Bangla <sup>ul</sup>
ar-DZ	Arabic <sup>ul</sup>	bo	Tibetan <sup>u</sup>
ar-MA	Arabic <sup>ul</sup>	brx	Bodo
ar-SY	Arabic <sup>ul</sup>	bs-Cyrl	Bosnian
as	Assamese	bs-Latn	Bosnian <sup>ul</sup>
asa	Asu	bs	Bosnian <sup>ul</sup>
ast	Asturian <sup>ul</sup>	ca	Catalan <sup>ul</sup>
az-Cyrl	Azerbaijani	ce	Chechen
az-Latn	Azerbaijani	cgg	Chiga
az	Azerbaijani <sup>ul</sup>	chr	Cherokee
bas	Basaa	ckb	Central Kurdish
be	Belarusian <sup>ul</sup>	cs	Czech <sup>ul</sup>

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

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

---

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

---

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

filipino  
finnish  
french-be  
french-belgium  
french-ca  
french-canada  
french-ch  
french-lu  
french-luxembourg  
french-switzerland  
french  
friulian  
fulah  
galician  
ganda  
georgian  
german-at  
german-austria  
german-ch  
german-switzerland  
german  
greek  
gujarati  
gusii  
hausa-gh  
hausa-ghana  
hausa-ne  
hausa-niger  
hausa  
hawaiian  
hebrew  
hindi  
hungarian  
icelandic  
igbo  
inarisami  
indonesian  
interlingua  
irish  
italian  
japanese  
jolafonyi  
kabuverdianu  
kabyle  
kako  
kalaallisut  
kalenjin  
kamba  
kannada  
kashmiri  
kazakh  
khmer  
kikuyu  
kinyarwanda

konkani  
korean  
koyraborosenni  
koyrachiini  
kwasio  
kyrgyz  
lakota  
langi  
lao  
latvian  
lingala  
lithuanian  
lowersorbian  
lsorbian  
lubakatanga  
luo  
luxembourgish  
luyia  
macedonian  
machame  
makhuwameetto  
makonde  
malagasy  
malay-bn  
malay-brunei  
malay-sg  
malay-singapore  
malay  
malayalam  
maltese  
manx  
marathi  
masai  
mazanderani  
meru  
meta  
mexican  
mongolian  
morisyen  
mundang  
nama  
nepali  
newzealand  
ngiemboon  
ngomba  
norsk  
northernluri  
northernsami  
northndebele  
norwegianbokmal  
norwegiannynorsk  
nswissgerman  
nuer  
nyankole



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

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

### 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=abcdefghijklj`. Keys may be added, too. Without `import` you may modify the identification keys. This can be used to create private variants easily. All you need is to import the same ini file with a different locale name and different parameters.

## 1.14 Selecting fonts

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

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

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

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

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

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

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

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

LUATEX/XETEX

```
\documentclass{article}

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

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

```

\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 $\langle lang \rangle$` , `\extras $\langle lang \rangle$` ) 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` [ $\langle options \rangle$ ] { $\langle language-name \rangle$ }

If the language  $\langle language-name \rangle$  has not been loaded as class or package option and there are no  $\langle options \rangle$ , 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`,  $\langle language-name \rangle$  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\maylangchaptername{..}
(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=** *<language-list>*

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

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

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

A special value is `+`, which allocates a new language (in the  $\text{\TeX}$  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, `Devanagari`). Overrides the value in the `ini` file. If `fontspec` does not define it, then `babel` sets its tag to that provided by the `ini` file. This value is particularly important because it sets the writing direction, so you must use it if for some reason the default value is wrong.

**language=**  $\langle$ *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 gereshe nor gershayim yet)  
**Hindi** alphabetic  
**Armenian** lower, upper  
**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, syllable, 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` `{<language>}{<true>}{<false>}`

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

`\localeinfo` `{<field>}`

**New 3.38** If an ini file has been loaded for the current language, you may access the information stored in it. This 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).

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`.

## 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` `*{\type}`  
`\babelhyphen` `*{\text}`

**New 3.9a** It is customary to classify hyphens in two types: (1) *explicit* or *hard hyphens*, which in T<sub>E</sub>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 T<sub>E</sub>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 T<sub>E</sub>X, `-` 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 L<sup>A</sup>T<sub>E</sub>X: (1) the character used is that set for the current font, while in L<sup>A</sup>T<sub>E</sub>X it is hardwired to `-` (a typical value); (2) the hyphen to be used in fonts with a negative `\hyphenchar` is `-`, like in L<sup>A</sup>T<sub>E</sub>X, 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>14</sup> adds or replaces patterns for the languages given or, without the optional argument, for *all* languages. If a pattern for a certain combination already exists, it gets replaced by the new one.

It can be used only in the preamble, and patterns are added when the language is first selected, thus taking into account changes of `\lccodes`'s done in `\extras<lang>` as well as the language-specific encoding (not set in the preamble by default). Multiple `\babelpatterns`'s are allowed.

Listed patterns are saved expanded and therefore it relies on the LICR. Of course, it also works without the LICR if the input and the font encodings are the same, like in Unicode based engines.

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

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

**`\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):

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

```

\babelposthyphenation{german}{([fmrtp]) | {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 ([ $\acute{u}$ ]), the replacement could be {1| $\acute{u}$ | $\acute{u}$ }, which maps  $\acute{u}$  to  $\acute{u}$ , and  $\acute{u}$  to  $\acute{u}$ , 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  $\check{z}$  as `zh` and  $\check{s}$  as `sh` in a newly created locale for transliterated Russian:

```

\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 Selecting scripts

Currently babel provides no standard interface to select scripts, because they are best selected with either `\fontencoding` (low-level) or a language name (high-level). Even the Latin script may require different encodings (ie, sets of glyphs) depending on the language, and therefore such a switch would be in a sense incomplete.<sup>15</sup>

Some languages sharing the same script define macros to switch it (eg, `\textcyrillic`), but be aware they may also set the language to a certain default. Even the babel core defined `\textlatin`, but it was somewhat buggy because in some cases it messed up encodings and fonts (for example, if the main Latin encoding was `LY1`), and therefore it has been deprecated.<sup>16</sup>

`\ensureascii` { $\langle text \rangle$ }

**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.

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

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

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.

## 1.21 Selecting directions

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

**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).

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

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

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

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

\end{document}
```

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

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

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

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

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

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

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

**counters** required in all engines (except `luatex` with `bidi=basic`) to reorder section numbers and the like (eg. `\subsection`..`\section`); required in `xetex` and `pdftex` for counters in general, as well as in `luatex` with `bidi=default`; required in `luatex` for numeric footnote marks  $>9$  with `bidi=basic-r` (but *not* with `bidi=basic`); note, however, it 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>17</sup>

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

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

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

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

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

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

**tabular** required in `luatex` for R tabular (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** .

<sup>17</sup>Next on the roadmap are counters and numeral systems in general. Expect some minor readjustments.



**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:

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

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

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

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

**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}{\footnote}%
\BabelFootnote{\localfootnote}{\language}{\footnote}%
\BabelFootnote{\mainfootnote}{\footnote}
```

(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}{\footnote}
```

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

## 1.22 Language attributes

### `\languageattribute`

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

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

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

## 1.23 Hooks

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

`\AddBabelHook` [`<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 `\useshortands*` to add a hook for the event `afterextras`). **New 3.33** They may be also applied to a specific language with the optional argument; language-specific settings are executed after global ones.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**loadkernel** (file) By default loads `switch.def`. It can be used to load a different version of this file or to load nothing.

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

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

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

## 1.24 Languages supported by babel with ldf files

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

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

## 1.25 Unicode character properties in luatex

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

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

**New 3.32** Here,  $\{ \langle \textit{char-code} \rangle \}$  is a number (with  $\text{\TeX}$  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 `\babelprovide`, or, if the last argument is empty, removes them. The last argument is the locale name:

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

## 1.26 Tweaking some features

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

**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 `luahbtex` you may need `bidi.mirroring=off`. Use with care, because these options do not deactivate other related options (like paragraph direction with `bidi.text`).

## 1.27 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`),  $\text{\LaTeX}$  will keep complaining about an undefined label. To prevent such problems, you could revert to using uppercase labels, you can use `\lowercase{\ref{foo}}` inside the argument of `\chapter`, or, if you will not use shorthands in labels, set the `safe` option to `none` or `bib`.

- Both `ltxdoc` and `babel` use `\AtBeginDocument` to change some catcodes, and `babel` reloads `hline` to make sure `:` has the right one, so if you want to change the catcode of `|` it has to be done using the same method at the proper place, with

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

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

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

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

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

- For the hyphenation to work correctly, `lccodes` cannot change, because  $\TeX$  only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.<sup>18</sup> So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of  $\TeX$ , not of `babel`. Alternatively, you may use `\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.
- `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`.

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

## 1.28 Current and future work

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

Useful additions would be, for example, time, currency, addresses and personal names.<sup>19</sup> But that is the easy part, because they don't require modifying the L<sup>A</sup>T<sub>E</sub>X internals. Calendars (Arabic, Persian, Indic, etc.) are under study.

Also interesting are differences in the sentence structure or related to it. For example, in Basque the number precedes the name (including chapters), in Hungarian “from (1)” is “(1)-ből”, but “from (3)” is “(3)-ből”, in Spanish an item labelled “3.<sup>o</sup>” may be referred to as either “ítem 3.<sup>o</sup>” or “3.<sup>er</sup> ítem”, and so on.

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

## 1.29 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:

- \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

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

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

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

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

3.9n, this task was delegated to the package `luatex-hyphen`, by Khaled Hosny, Élie Roux, and Manuel Pégourié-Gonnard, and required an extra file named `language.dat.lua`, but now a new mechanism has been devised based solely on `language.dat`. **You must rebuild the formats** if upgrading from a previous version. You may want to have a local `language.dat` for a particular project (for example, a book on Chemistry).<sup>21</sup>

## 2.1 Format

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

The file can contain empty lines and comments, as well as lines which start with an equals (=) sign. Such a line will instruct  $\LaTeX$  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>23</sup> For example:

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

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

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

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

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

## 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

<sup>21</sup>The loader for `lua(e)tex` is slightly different as it's not based on `babel` but on `etex.src`. Until 3.9p it just didn't work, but thanks to the new code it works by reloading the data in the `babel` way, i. e., with `language.dat`.

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

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

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 `\langle lang \rangle hyphenmins`, `\captions\langle lang \rangle`, `\date\langle lang \rangle`, `\extras\langle lang \rangle` and `\noextras\langle lang \rangle` (the last two may be left empty); where `\langle lang \rangle` is either the name of the language definition file or the name of the  $\text{\LaTeX}$  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\langle lang \rangle` but not `\captions\langle lang \rangle` does not raise an error but can lead to unexpected results.
- When a language definition file is loaded, it can define `\l@\langle lang \rangle` to be a dialect of `\language0` when `\l@\langle lang \rangle` is undefined.
- Language names must be all lowercase. If an unknown language is selected, babel will attempt setting it after lowercasing its name.
- The semantics of modifiers is not defined (on purpose). In most cases, they will just be simple separated options (eg, spanish), but a language might require, say, a set of options organized as a tree with suboptions (in such a case, the recommended separator is `/`).

Some recommendations:

- The preferred shorthand is `"`, which is not used in  $\text{\LaTeX}$  (quotes are entered as ``` and `'`). Other good choices are characters which are not used in a certain context (eg, `=` in an ancient language). Note however `=`, `<`, `>`, `:` and the like can be dangerous, because they may be used as part of the syntax of some elements (numeric expressions, key/value pairs, etc.).
- Captions should not contain shorthands or encoding-dependent commands (the latter is not always possible, but should be clearly documented). They should be defined using the LICR. You may also use the new tools for encoded strings, described below.
- Avoid adding things to `\noextras\langle lang \rangle` 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\langle lang \rangle`.
- Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the font encoding (low-level) or the language (high-level, which in turn may switch the font encoding). Usage of things like `\latintext` is deprecated.<sup>24</sup>
- Please, for “private” internal macros do not use the `\bbl@` prefix. It is used by babel and it can lead to incompatibilities.

---

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



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

### 3.1 Guidelines for contributed languages

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

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

The following page provides a starting point: <http://www.texnia.com/incubator.html>. If you need further assistance and technical advice in the development of language styles, I am willing to help you. And of course, you can make any suggestion you like.

### 3.2 Basic macros

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

`\addlanguage` The macro `\addlanguage` is a non-outer version of the macro `\newlanguage`, defined in `plain.tex` version 3.x. For older versions of `plain.tex` and `lplain.tex` a substitute definition is used. Here “language” is used in the TeX sense of set of hyphenation patterns.

`\adddialect` The macro `\adddialect` can be used when two languages can (or must) use the same hyphenation patterns. This can also be useful for languages for which no patterns are preloaded in the format. In such cases the default 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 <i>not</i> set them).
<code>\captions&lt;lang&gt;</code>	The macro <code>\captions&lt;lang&gt;</code> defines the macros that hold the texts to replace the original hard-wired texts.
<code>\date&lt;lang&gt;</code>	The macro <code>\date&lt;lang&gt;</code> defines <code>\today</code> .
<code>\extras&lt;lang&gt;</code>	The macro <code>\extras&lt;lang&gt;</code> contains all the extra definitions needed for a specific language. This macro, like the following, is a hook – you can add things to it, but it must not be used directly.
<code>\noextras&lt;lang&gt;</code>	Because we want to let the user switch between languages, but we do not know what state $\TeX$ might be in after the execution of <code>\extras&lt;lang&gt;</code> , a macro that brings $\TeX$ into a predefined state is needed. It will be no surprise that the name of this macro is <code>\noextras&lt;lang&gt;</code> .
<code>\bbl@declareattribute</code>	This is a command to be used in the language definition files for declaring a language attribute. It takes three arguments: the name of the language, the attribute to be defined, and the code to be executed when the attribute is to be used.
<code>\main@language</code>	To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use <code>\main@language</code> instead of <code>\selectlanguage</code> . This will just store the name of the language, and the proper language will be activated at the start of the document.
<code>\ProvidesLanguage</code>	The macro <code>\ProvidesLanguage</code> should be used to identify the language definition files. Its syntax is similar to the syntax of the $\TeX$ command <code>\ProvidesPackage</code> .
<code>\LdfInit</code>	The macro <code>\LdfInit</code> performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the <code>@</code> -sign, preventing the <code>.ldf</code> file from being processed twice, etc.
<code>\ldf@quit</code>	The macro <code>\ldf@quit</code> does work needed if a <code>.ldf</code> file was processed earlier. This includes resetting the category code of the <code>@</code> -sign, preparing the language to be activated at <code>\begin{document}</code> time, and ending the input stream.
<code>\ldf@finish</code>	The macro <code>\ldf@finish</code> does work needed at the end of each <code>.ldf</code> file. This includes resetting the category code of the <code>@</code> -sign, loading a local configuration file, and preparing the language to be activated at <code>\begin{document}</code> time.
<code>\loadlocalcfg</code>	After processing a language definition file, $\TeX$ can be instructed to load a local configuration file. This file can, for instance, be used to add strings to <code>\captions&lt;lang&gt;</code> to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by <code>\ldf@finish</code> .
<code>\substitutefontfamily</code>	(Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This <code>.fd</code> file will instruct $\TeX$ to use a font from the second family when a font from the first family in the given encoding seems to be needed.

### 3.3 Skeleton

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

```

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

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

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

```

```

\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`.  $\LaTeX$  adds another macro called `\@sanitize` representing the same character set, but without the curly braces. The macros `\bbl@add@special<char>` and `\bbl@remove@special<char>` add and remove the character `<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>25</sup>.

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

`\babel@savevariable` A second macro is provided to save the current value of a variable. In this context, anything that is allowed after the `\the` primitive is considered to be a variable. The macro takes one argument, the `<variable>`.  
 The effect of the preceding macros is to append a piece of code to the current definition of `\originalTeX`. When `\originalTeX` is expanded, this code restores the previous definition of the control sequence or the previous value of the variable.

### 3.6 Support for extending macros

`\addto` The macro `\addto{<control sequence>}{<TeX code>}` can be used to extend the definition of a macro. The macro need not be defined (ie, it can be undefined or `\relax`). This macro can, for instance, be used in adding instructions to a macro like `\extrasenglish`. Be careful when using this macro, because depending on the case the assignment 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

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

`\set@low@box` is available. It takes one argument and puts that argument in an `\hbox`, at the baseline. The result is available in `\box0` for further processing.

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

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

`\bbl@nonfrenchspacing`

### 3.8 Encoding-dependent strings

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

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

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

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

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

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

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

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

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

The  $\langle category \rangle$  is either captions, date or extras. You must stick to these three categories, even if no error is raised when using other name.<sup>26</sup> It may be empty, too, but in such a case using  $\backslash SetString$  is an error (but not  $\backslash SetCase$ ).

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

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

\EndBabelCommands
```

A real example is:

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

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

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

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

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

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

\EndBabelCommands
```

When used in ldf files, previous values of  $\backslash \langle category \rangle \langle language \rangle$  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).

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

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

The starred version just forces strings to take a value – if not set as package option, then the default for the engine is used. This is not done by default to prevent backward incompatibilities, but if you are creating a new language this version is better. It's up to the maintainers of the current languages to decide if using it is appropriate.<sup>27</sup>

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

**\AfterBabelCommands** `{\langle code \rangle}`

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

**\SetString** `{\langle macro-name \rangle}{\langle string \rangle}`

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

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

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

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

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

#1 is replaced by the roman numeral.

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

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

\StartBabelCommands{turkish}{}[
```

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



```

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

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

```

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

```

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

## 4 Changes

### 4.1 Changes in babel version 3.9

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

- `\select@language` did not set `\language`name. This meant the language in force when auxiliary files were loaded was the one used in, for example, shorthands – if the language was `german`, a `\select@language{spanish}` had no effect.
- `\foreignlanguage` and `otherlanguage*` messed up `\extras<language>`. Scripts, encodings and many other things were not switched correctly.



- The :ENC mechanism for hyphenation patterns used the encoding of the *previous* language, not that of the language being selected.
- ' (with activeacute) had the original value when writing to an auxiliary file, and things like an infinite loop could happen. It worked incorrectly with ^ (if activated) and also if deactivated.
- Active chars were not reset at the end of language options, and that led to incompatibilities between languages.
- \textormath raised an error with a conditional.
- \aliasshorthand didn't work (or only in a few and very specific cases).
- \l@english was defined incorrectly (using \let instead of \chardef).
- ldf files not bundled with babel were not recognized when called as global options.

## 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 on <http://tug.org/mailman/listinfo/kadingira>).

## 5 Identification and loading of required files

*Code documentation is still under revision.*

The babel package after unpacking consists of the following files:

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

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

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

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

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

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

## 6 locale directory

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

Most of them are essentially finished (except bugs and mistakes, of course). Some of them are still incomplete (but they will be usable), and there are some omissions (eg, Latin and

polytonic Greek, and there are no geographic areas in Spanish). Hindi, French, Occitan and Breton will show a warning related to dates. Not all include LICR variants.

This is a preliminary documentation.

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

Most keys are self-explanatory.

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

**version** of the ini file

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

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

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

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

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

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

## 7 Tools

```
1 <<version=3.41.1952>>
2 <<date=2020/03/20>>
```

**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 L<sup>A</sup>T<sub>E</sub>X 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` Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the `\else` and `\fi` parts of an `\if`-statement<sup>28</sup>. These macros will break if another `\if... \fi` statement appears in one of the arguments and it is not enclosed in braces.

`\bbl@afterfi`

```
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}
```

`\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##2 \@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
```

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

```

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 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>>

```

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

```

160 <<*Load patterns in luatex>> ≡
161 \ifx\directlua\@undefined\else
162 \ifx\bbl@luapatterns\@undefined
163 \input luabel.def
164 \fi
165 \fi
166 <</Load patterns in luatex>>

```

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

```

167 <<*Load macros for plain if not LaTeX>> ≡
168 \ifx\AtBeginDocument\@undefined
169 \input plain.def\relax
170 \fi
171 <</Load macros for plain if not LaTeX>>

```

## 7.1 Multiple languages

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

172 <<*Define core switching macros>> ≡
173 \ifx\language\@undefined
174 \csname newcount\endcsname\language
175 \fi
176 <</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  $\TeX$ 's memory plain  $\TeX$  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  $\TeX$  version 3.0.

For formats based on plain version 3.0 the definition of `\newlanguage` can be simply copied, removing `\outer`. Plain  $\TeX$  version 3.0 uses `\count 19` for this purpose.

```

177 <<*Define core switching macros>> ≡
178 \ifx\newlanguage\undefined
179   \csname newcount\endcsname\last@language
180   \def\addlanguage#1{%
181     \global\advance\last@language\@ne
182     \ifnum\last@language<\@ccclvi
183       \else
184         \errmessage{No room for a new \string\language!}%
185       \fi
186     \global\chardef#1\last@language
187     \wlog{\string#1 = \string\language\the\last@language}}
188   \else
189     \countdef\last@language=19
190     \def\addlanguage{\alloc@9\language\chardef\@ccclvi}
191   \fi
192 <</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  $\TeX$  2.09. In that case the file `plain.def` is needed (which also defines `\AtBeginDocument`, and therefore it is not loaded twice). We need the first part when the format is created, and `\orig@dump` is used as a flag. Otherwise, we need to use the second part, so `\orig@dump` is not defined (`plain.def` undefines it).

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

## 8 The Package File ( $\TeX$ , `babel.sty`)

In order to make use of the features of  $\TeX$  2<sub>ε</sub>, the `babel` system contains a package file, `babel.sty`. This file is loaded by the `\usepackage` command and defines all the language options whose name is different from that of the `.ldf` file (like variant spellings). It also takes care of a number of compatibility issues with other packages and defines a few additional package options.

Apart from all the language options below we also have a few options that influence the behavior of language definition files.

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

### 8.1 base

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

```

193 (*package)
194 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
195 \ProvidesPackage{babel}[\<<date>> \<<version>> The Babel package]
196 \ifpackagewith{babel}{debug}
197   {\providecommand\bb1@trace[1]{\message{^^J[ #1 ]}}}%

```

```

198 \let\bbl@debug\@firstofone}
199 {\providecommand\bbl@trace[1]{}%
200 \let\bbl@debug@gobble}
201 \ifx\bbl@switchflag\undefined % Prevent double input
202 \let\bbl@switchflag\relax
203 \input switch.def\relax
204 \fi
205 <<Load patterns in luatex>>
206 <<Basic macros>>
207 \def\AfterBabelLanguage#1{%
208 \global\expandafter\bbl@add\csname#1.ldf-h@k\endcsname}%

```

If the format created a list of loaded languages (in \bbl@languages), get the name of the 0-th to show the actual language used.

```

209 \ifx\bbl@languages\undefined\else
210 \begingroup
211 \catcode\^^I=12
212 \@ifpackagewith{babel}{showlanguages}{%
213 \begingroup
214 \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
215 \wlog{<*languages>}%
216 \bbl@languages
217 \wlog{</languages>}%
218 \endgroup}{%
219 \endgroup
220 \def\bbl@elt#1#2#3#4{%
221 \ifnum#2=\z@
222 \gdef\bbl@nulllanguage{#1}%
223 \def\bbl@elt##1##2##3##4{}}%
224 \fi}%
225 \bbl@languages
226 \fi
227 \ifodd\bbl@engine
228 \def\bbl@activate@preotf{%
229 \let\bbl@activate@preotf\relax % only once
230 \directlua{
231 Babel = Babel or {}
232 %
233 function Babel.pre_otfload_v(head)
234 if Babel.numbers and Babel.digits_mapped then
235 head = Babel.numbers(head)
236 end
237 if Babel.bidi_enabled then
238 head = Babel.bidi(head, false, dir)
239 end
240 return head
241 end
242 %
243 function Babel.pre_otfload_h(head, gc, sz, pt, dir)
244 if Babel.numbers and Babel.digits_mapped then
245 head = Babel.numbers(head)
246 end
247 if Babel.bidi_enabled then
248 head = Babel.bidi(head, false, dir)
249 end
250 return head
251 end
252 %
253 luatexbase.add_to_callback('pre_linebreak_filter',

```



```

254     Babel.pre_otfload_v,
255     'Babel.pre_otfload_v',
256     luatexbase.priority_in_callback('pre_linebreak_filter',
257     'luaotfload.node_processor') or nil)
258 %
259     luatexbase.add_to_callback('hpack_filter',
260     Babel.pre_otfload_h,
261     'Babel.pre_otfload_h',
262     luatexbase.priority_in_callback('hpack_filter',
263     'luaotfload.node_processor') or nil)
264 }}
265 \let\bbl@tempa\relax
266 \@ifpackagewith{babel}{bidi=basic}%
267 {\def\bbl@tempa{basic}}%
268 {\@ifpackagewith{babel}{bidi=basic-r}%
269 {\def\bbl@tempa{basic-r}}%
270 {}}
271 \ifx\bbl@tempa\relax\else
272 \let\bbl@beforeforeign\leavevmode
273 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}%
274 \RequirePackage{luatexbase}%
275 \directlua{
276     require('babel-data-bidi.lua')
277     require('babel-bidi-\bbl@tempa.lua')
278 }
279 \bbl@activate@preotf
280 \fi
281 \fi

```

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

```

282 \bbl@trace{Defining option 'base'}
283 \@ifpackagewith{babel}{base}{%
284 \ifx\directlua\undefined
285 \DeclareOption*{\bbl@patterns{\CurrentOption}}%
286 \else
287 \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
288 \fi
289 \DeclareOption{base}{}%
290 \DeclareOption{showlanguages}{}%
291 \ProcessOptions
292 \global\expandafter\let\csname opt@babel.sty\endcsname\relax
293 \global\expandafter\let\csname ver@babel.sty\endcsname\relax
294 \global\let@ifl@ter@@\@ifl@ter
295 \def\@ifl@ter#1#2#3#4#5{\global\let@ifl@ter\@ifl@ter@@}%
296 \endinput}{}%

```

## 8.2 key=value options and other general option

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

```

297 \bbl@trace{key=value and another general options}
298 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
299 \def\bbl@tempb#1.#2{%
300     #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%

```

```

301 \def\bbl@tempd#1.#2\@nnil{%
302   \ifx\@empty#2%
303     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
304   \else
305     \in{@}{#1}\ifin@
306     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
307   \else
308     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
309     \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
310   \fi
311 \fi}
312 \let\bbl@tempc\@empty
313 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
314 \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.

```

315 \DeclareOption{KeepShorthandsActive}{}
316 \DeclareOption{activeacute}{}
317 \DeclareOption{activegrave}{}
318 \DeclareOption{debug}{}
319 \DeclareOption{noconfigs}{}
320 \DeclareOption{showlanguages}{}
321 \DeclareOption{silent}{}
322 \DeclareOption{mono}{}
323 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
324 % Don't use. Experimental:
325 \newif\ifbbl@single
326 \DeclareOption{selectors=off}{\bbl@singletrue}
327 <<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.

```

328 \let\bbl@opt@shorthands\@nnil
329 \let\bbl@opt@config\@nnil
330 \let\bbl@opt@main\@nnil
331 \let\bbl@opt@headfoot\@nnil
332 \let\bbl@opt@layout\@nnil

```

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

```

333 \def\bbl@tempa#1=#2\bbl@tempa{%
334   \bbl@csarg\ifx{opt@#1}\@nnil
335     \bbl@csarg\edef{opt@#1}{#2}%
336   \else
337     \bbl@error{%
338       Bad option `#1=#2'. Either you have misspelled the\\%
339       key or there is a previous setting of `#1'}{%
340       Valid keys are `shorthands', `config', `strings', `main',\\%
341       `headfoot', `safe', `math', among others.}
342   \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.

```

343 \let\bbl@language@opts\@empty
344 \DeclareOption*{%
345   \bbl@xin@\string=\CurrentOption}%
346   \ifin@
347     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
348   \else
349     \bbl@add@list\bbl@language@opts{\CurrentOption}%
350   \fi}

```

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

```

351 \ProcessOptions*

```

### 8.3 Conditional loading of shorthands

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

```

352 \bbl@trace{Conditional loading of shorthands}
353 \def\bbl@sh@string#1{%
354   \ifx#1\@empty\else
355     \ifx#1t\string~%
356     \else\ifx#1c\string,%
357     \else\string#1%
358   \fi\fi
359   \expandafter\bbl@sh@string
360 \fi}
361 \ifx\bbl@opt@shorthands\@nnil
362   \def\bbl@ifshorthand#1#2#3{#2}%
363 \else\ifx\bbl@opt@shorthands\@empty
364   \def\bbl@ifshorthand#1#2#3{#3}%
365 \else

```

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

```

366 \def\bbl@ifshorthand#1{%
367   \bbl@xin@\string#1}{\bbl@opt@shorthands}%
368   \ifin@
369     \expandafter\@firstoftwo
370   \else
371     \expandafter\@secondoftwo
372   \fi}

```

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

```

373 \edef\bbl@opt@shorthands{%
374   \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.

```

375 \bbl@ifshorthand{'}%
376   {\PassOptionsToPackage{activeacute}{babel}}{}
377 \bbl@ifshorthand{`}%
378   {\PassOptionsToPackage{activegrave}{babel}}{}
379 \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.

```
380 \ifx\bbl@opt@headfoot\@nnil\else
381   \g@addto@macro\@resetactivechars{%
382     \set@typeset@protect
383     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
384     \let\protect\noexpand}
385 \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.

```
386 \ifx\bbl@opt@safe\@undefined
387   \def\bbl@opt@safe{BR}
388 \fi
389 \ifx\bbl@opt@main\@nnil\else
390   \edef\bbl@language@opts{%
391     \ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
392     \bbl@opt@main}
393 \fi
```

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

```
394 \bbl@trace{Defining IfBabelLayout}
395 \ifx\bbl@opt@layout\@nnil
396   \newcommand\IfBabelLayout[3]{#3}%
397 \else
398   \newcommand\IfBabelLayout[1]{%
399     \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
400     \ifin@
401       \expandafter\@firstoftwo
402     \else
403       \expandafter\@secondoftwo
404     \fi}
405 \fi
```

## 8.4 Language options

Languages are loaded when processing the corresponding option *except* if a main language has been set. In such a case, it is not loaded until all options has been processed. The following macro inputs the `ldf` file and does some additional checks (`\input` works, too, but possible errors are not caught).

```
406 \bbl@trace{Language options}
407 \let\bbl@afterlang\relax
408 \let\BabelModifiers\relax
409 \let\bbl@loaded\@empty
410 \def\bbl@load@language#1{%
411   \InputIfFileExists{#1.ldf}%
412   {\edef\bbl@loaded{\CurrentOption
413     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
414     \expandafter\let\expandafter\bbl@afterlang
415       \csname\CurrentOption.ldf-h@@k\endcsname
416     \expandafter\let\expandafter\BabelModifiers
417       \csname bbl@mod@\CurrentOption\endcsname}%
418   {\bbl@error{%
419     Unknown option '\CurrentOption'. Either you misspelled it\\%
420     or the language definition file \CurrentOption.ldf was not found}}%
421   Valid options are: shorthands=, KeepShorthandsActive,\\%
```

```

422     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
423     headfoot=, strings=, config=, hyphenmap=, or a language name.}}

```

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

```

424 \def\bbl@try@load@lang#1#2#3{%
425     \IfFileExists{\CurrentOption.lda}%
426     {\bbl@load@language{\CurrentOption}}%
427     {\#1\bbl@load@language{\#2}\#3}}
428 \DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}{}}
429 \DeclareOption{brazil}{\bbl@try@load@lang{}{portuges}{}}
430 \DeclareOption{brazilian}{\bbl@try@load@lang{}{portuges}{}}
431 \DeclareOption{hebrew}{%
432     \input{rlbabel.def}%
433     \bbl@load@language{hebrew}}
434 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
435 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
436 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
437 \DeclareOption{polutonikogreek}{%
438     \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
439 \DeclareOption{portuguese}{\bbl@try@load@lang{}{portuges}{}}
440 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
441 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
442 \DeclareOption{uppersorbian}{\bbl@try@load@lang{}{usorbian}{}}

```

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

```

443 \ifx\bbl@opt@config\@nnil
444     \@ifpackagewith{babel}{noconfigs}{}%
445     {\InputIfFileExists{bblopts.cfg}%
446         {\typeout{*****^J%
447             * Local config file bblopts.cfg used^^J%
448             *}}%
449         {}}%
450 \else
451     \InputIfFileExists{\bbl@opt@config.cfg}%
452     {\typeout{*****^J%
453         * Local config file \bbl@opt@config.cfg used^^J%
454         *}}%
455     {\bbl@error{%
456         Local config file '\bbl@opt@config.cfg' not found}{%
457         Perhaps you misspelled it.}}%
458 \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.

```

459 \bbl@for\bbl@tempa\bbl@language@opts{%
460     \bbl@ifunset{ds@\bbl@tempa}%
461     {\edef\bbl@tempb{%
462         \noexpand\DeclareOption
463         {\bbl@tempa}%
464         {\noexpand\bbl@load@language{\bbl@tempa}}}%
465     \bbl@tempb}%
466     \@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.

```

467 \bbl@foreach\@classoptionslist{%
468   \bbl@ifunset{ds@#1}%
469   {\IfFileExists{#1.ldf}%
470    {\DeclareOption{#1}{\bbl@load@language{#1}}}%
471    {}}%
472  {}}

```

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

```

473 \ifx\bbl@opt@main\@nnil\else
474   \expandafter
475   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
476   \DeclareOption{\bbl@opt@main}{}
477 \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):

```

478 \def\AfterBabelLanguage#1{%
479   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
480 \DeclareOption*{}
481 \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`.

```

482 \ifx\bbl@opt@main\@nnil
483   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
484   \let\bbl@tempc\@empty
485   \bbl@for\bbl@tempb\bbl@tempa{%
486     \bbl@xin@{\bbl@tempb,}{,\bbl@loaded,}%
487     \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
488   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
489   \expandafter\bbl@tempa\bbl@loaded,\@nnil
490   \ifx\bbl@tempb\bbl@tempc\else
491     \bbl@warning{%
492       Last declared language option is '\bbl@tempc',\%
493       but the last processed one was '\bbl@tempb'.\%
494       The main language cannot be set as both a global\%
495       and a package option. Use 'main=\bbl@tempc' as\%
496       option. Reported}%
497   \fi
498 \else
499   \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
500   \ExecuteOptions{\bbl@opt@main}
501   \DeclareOption*{}
502   \ProcessOptions*
503 \fi
504 \def\AfterBabelLanguage{%
505   \bbl@error
506   {Too late for \string\AfterBabelLanguage}%
507   {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.

```

508 \ifx\bbl@main@language\@undefined
509   \bbl@info{%
510     You haven't specified a language. I'll use 'nil'\%
511     as the main language. Reported}
512   \bbl@load@language{nil}
513 \fi
514 \</package>
515 \*core>

```

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

The kernel of the babel system is stored in either `hyphen.cfg` or `switch.def` and `babel.def`. The file `babel.def` contains most of the code, while `switch.def` defines the language-switching commands; both can be read at run time. The file `hyphen.cfg` is a file that can be loaded into the format, which is necessary when you want to be able to switch hyphenation patterns (by default, it also inputs `switch.def`, for “historical reasons”, but it is not necessary). When `babel.def` is loaded it checks if the current version of `switch.def` is in the format; if not, it is loaded. A further file, `babel.sty`, contains  $\text{\LaTeX}$ -specific stuff. Because plain  $\text{\TeX}$  users might want to use some of the features of the babel system too, care has to be taken that plain  $\text{\TeX}$  can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain  $\text{\TeX}$  and  $\text{\LaTeX}$ , some of it is for the  $\text{\LaTeX}$  case only.

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

### 9.1 Tools

```

516 \ifx\ldf@quit\@undefined
517 \else
518   \expandafter\endinput
519 \fi
520 <<Make sure ProvidesFile is defined>>
521 \ProvidesFile{babel.def}[\<date>] <<version>> Babel common definitions]
522 <<Load macros for plain if not LaTeX>>

```

The file `babel.def` expects some definitions made in the  $\text{\LaTeX} 2_{\epsilon}$  style file. So, In  $\text{\LaTeX} 2.09$  and Plain we must provide at least some predefined values as well some tools to set them (even if not all options are available). There are no package options, and therefore an alternative mechanism is provided. For the moment, only `\babeloptionstrings` and `\babeloptionmath` are provided, which can be defined before loading babel.

`\BabelModifiers` can be set too (but not sure it works).

```

523 \ifx\bbl@ifshorthand\@undefined
524   \let\bbl@opt@shorthands\@nnil
525   \def\bbl@ifshorthand#1#2#3{#2}%
526   \let\bbl@language@opts\@empty
527   \ifx\babeloptionstrings\@undefined
528     \let\bbl@opt@strings\@nnil
529   \else
530     \let\bbl@opt@strings\babeloptionstrings
531   \fi
532   \def\BabelStringsDefault{generic}
533   \def\bbl@tempa{normal}

```

```

534 \ifx\babeloptionmath\bb1@tempa
535 \def\bb1@mathnormal{\noexpand\textormath}
536 \fi
537 \def\AfterBabelLanguage#1#2{
538 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
539 \let\bb1@afterlang\relax
540 \def\bb1@opt@safe{BR}
541 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
542 \ifx\bb1@trace\@undefined\def\bb1@trace#1{}\fi
543 \expandafter\newif\csname ifbb1@single\endcsname
544 \fi

And continue.
545 \ifx\bb1@switchflag\@undefined % Prevent double input
546 \let\bb1@switchflag\relax
547 \input switch.def\relax
548 \fi
549 \bb1@trace{Compatibility with language.def}
550 \ifx\bb1@languages\@undefined
551 \ifx\directlua\@undefined
552 \openin1 = language.def
553 \ifeof1
554 \closein1
555 \message{I couldn't find the file language.def}
556 \else
557 \closein1
558 \begingroup
559 \def\addlanguage#1#2#3#4#5{%
560 \expandafter\ifx\csname lang@#1\endcsname\relax\else
561 \global\expandafter\let\csname l@#1\expandafter\endcsname
562 \csname lang@#1\endcsname
563 \fi}%
564 \def\uselanguage#1{%
565 \input language.def
566 \endgroup
567 \fi
568 \fi
569 \chardef\l@english\z@
570 \fi
571 <<Load patterns in luatex>>
572 <<Basic macros>>

```

\addto For each language four control sequences have to be defined that control the language-specific definitions. To be able to add something to these macro once they have been defined the macro \addto is introduced. It takes two arguments, a *<control sequence>* and T<sub>E</sub>X-code to be added to the *<control sequence>*.

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

```

573 \def\addto#1#2{%
574 \ifx#1\@undefined
575 \def#1{#2}%
576 \else
577 \ifx#1\relax
578 \def#1{#2}%
579 \else
580 {\toks@\expandafter{#1#2}%

```



```

581 \xdef#1{\the\toks@}}%
582 \fi
583 \fi}

```

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

```

584 \def\bbl@withactive#1#2{%
585 \begingroup
586 \lccode`~=#2\relax
587 \lowercase{\endgroup#1~}}

```

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

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

```

588 \def\bbl@redefine#1{%
589 \edef\bbl@tempa{\bbl@stripslash#1}%
590 \expandafter\let\csname org@\bbl@tempa\endcsname#1%
591 \expandafter\def\csname\bbl@tempa\endcsname}

```

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

```

592 \@onlypreamble\bbl@redefine

```

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

```

593 \def\bbl@redefine@long#1{%
594 \edef\bbl@tempa{\bbl@stripslash#1}%
595 \expandafter\let\csname org@\bbl@tempa\endcsname#1%
596 \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
597 \@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`.

```

598 \def\bbl@redefineroobust#1{%
599 \edef\bbl@tempa{\bbl@stripslash#1}%
600 \bbl@ifunset{\bbl@tempa\space}%
601 {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
602 \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
603 {\bbl@exp{\let\<org@\bbl@tempa\<\bbl@tempa\space>}}}%
604 \@namedef{\bbl@tempa\space}}

```

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

```

605 \@onlypreamble\bbl@redefineroobust

```

## 9.2 Hooks

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

```

606 \bbl@trace{Hooks}

```

```

607 \newcommand\AddBabelHook[3][]{%
608   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
609   \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
610   \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
611   \bbl@ifunset{bbl@ev@#2@#3@#1}%
612   {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elt{#2}}}%
613   {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
614   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
615 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
616 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
617 \def\bbl@usehooks#1#2{%
618   \def\bbl@elt##1{%
619     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@#2}}%
620     \bbl@cs{ev@#1@}%
621     \ifx\language\@undefined\else % Test required for Plain (?)
622       \def\bbl@elt##1{%
623         \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1@#2}}%
624         \bbl@cl{ev@#1}%
625       \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).

```

626 \def\bbl@evargs{,% <- don't delete this comma
627   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
628   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
629   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
630   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
631   beforestart=0,language=0}

```

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

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

```

632 \bbl@trace{Defining babelensure}
633 \newcommand\babelensure[2][]{% TODO - revise test files
634   \AddBabelHook{babel-ensure}{afterextras}{%
635     \ifcase\bbl@select@type
636       \bbl@cl{e}%
637     \fi}%
638   \begingroup
639     \let\bbl@ens@include\@empty
640     \let\bbl@ens@exclude\@empty
641     \def\bbl@ens@fontenc{\relax}%
642     \def\bbl@tempb##1{%
643       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
644     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
645     \def\bbl@tempb##1=##2\@{\@namedef{bbl@ens@##1}{##2}}%
646     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
647     \def\bbl@tempc{\bbl@ensure}%
648     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%

```

```

649 \expandafter{\bbl@ens@include}}%
650 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
651 \expandafter{\bbl@ens@exclude}}%
652 \toks@\expandafter{\bbl@tempc}%
653 \bbl@exp{%
654 \endgroup
655 \def<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
656 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
657 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
658 \ifx##1\@undefined % 3.32 - Don't assume the macros exists
659 \edef##1{\noexpand\bbl@nocaption
660 {\bbl@stripslash##1}{\language\bbl@stripslash##1}}%
661 \fi
662 \ifx##1\@empty\else
663 \in@{##1}{#2}%
664 \ifin@else
665 \bbl@ifunset{\bbl@ensure@\language}%
666 {\bbl@exp{%
667 \\\DeclareRobustCommand<bbl@ensure@\language>[1]{%
668 \\\foreignlanguage{\language}%
669 {\ifx\relax#3\else
670 \\\fontencoding{#3}\selectfont
671 \fi
672 #####1}}}%
673 }%
674 \toks@\expandafter{##1}%
675 \edef##1{%
676 \bbl@csarg\noexpand{ensure@\language}%
677 {\the\toks@}}%
678 \fi
679 \expandafter\bbl@tempb
680 \fi}%
681 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
682 \def\bbl@tempa##1{% elt for include list
683 \ifx##1\@empty\else
684 \bbl@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
685 \ifin@else
686 \bbl@tempb##1\@empty
687 \fi
688 \expandafter\bbl@tempa
689 \fi}%
690 \bbl@tempa#1\@empty}
691 \def\bbl@captionslist{%
692 \prefacename\refname\abstractname\bibname\chaptername\appendixname
693 \contentsname\listfigurename\listtablename\indexname\figurename
694 \tablename\partname\enclname\ccname\headtoname\pagename\seename
695 \alsoname\proofname\glossaryname}

```

### 9.3 Setting up language files

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

At the start of processing a language definition file we always check the category code of the 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`.

```

696 \bbl@trace{Macros for setting language files up}
697 \def\bbl@ldfinit{%
698   \let\bbl@screset\@empty
699   \let\BabelStrings\bbl@opt@string
700   \let\BabelOptions\@empty
701   \let\BabelLanguages\relax
702   \ifx\originalTeX\@undefined
703     \let\originalTeX\@empty
704   \else
705     \originalTeX
706   \fi}
707 \def\LdfInit#1#2{%
708   \chardef\atcatcode=\catcode`\@
709   \catcode`\@=11\relax
710   \chardef\eqcatcode=\catcode`\=
711   \catcode`\==12\relax
712   \expandafter\if\expandafter\@backslashchar
713     \expandafter\@car\string#2\@nil
714   \ifx#2\@undefined\else
715     \ldf@quit{#1}%
716   \fi
717 \else
718   \expandafter\ifx\csname#2\endcsname\relax\else
719     \ldf@quit{#1}%
720   \fi
721 \fi
722 \bbl@ldfinit}

```

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

```

723 \def\ldf@quit#1{%
724   \expandafter\main@language\expandafter{#1}%
725   \catcode`\@=\atcatcode \let\atcatcode\relax
726   \catcode`\==\eqcatcode \let\eqcatcode\relax
727   \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.

```

728 \def\bbl@afterldf#1{%
729   \bbl@afterlang
730   \let\bbl@afterlang\relax
731   \let\BabelModifiers\relax

```

```

732 \let\bbl@screaset\relax}%
733 \def\ldf@finish#1{%
734 \loadlocalcfg{#1}%
735 \bbl@afterldf{#1}%
736 \expandafter\main@language\expandafter{#1}%
737 \catcode\@=\atcatcode \let\atcatcode\relax
738 \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  $\LaTeX$ .

```

739 \@onlypreamble\LdfInit
740 \@onlypreamble\ldf@quit
741 \@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.

```

742 \def\main@language#1{%
743 \def\bbl@main@language{#1}%
744 \let\language\main@language
745 \bbl@id@assign
746 \bbl@patterns{\language}

```

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

```

747 \def\bbl@beforestart{%
748 \bbl@usehooks{beforestart}}}%
749 \global\let\bbl@beforestart\relax}
750 \AtBeginDocument{%
751 \bbl@cs{beforestart}%
752 \if@filesw
753 \immediate\write\@mainaux{\string\bbl@cs{beforestart}}%
754 \fi
755 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
756 \ifbbl@single % must go after the line above
757 \renewcommand\selectlanguage[1]{}%
758 \renewcommand\foreignlanguage[2]{#2}%
759 \global\let\babel@aux\@gobbles % Also as flag
760 \fi
761 \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

762 \def\select@language@x#1{%
763 \ifcase\bbl@select@type
764 \bbl@ifsamestring\language{#1}{\select@language{#1}}%
765 \else
766 \select@language{#1}%
767 \fi}

```

## 9.4 Shorthands

`\bbl@add@special` The macro `\bbl@add@special` is used to add a new character (or single character control sequence) to the macro `\dospecials` (and `\@sanitize` if  $\LaTeX$  is used). It is used only at one place, namely when `\initiate@active@char` is called (which is ignored if the char has been made active before). Because `\@sanitize` can be undefined, we put the definition inside a conditional.

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

```

768 \bbl@trace{Shorhands}
769 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
770 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
771 \bbl@ifunset{@sanitize}{\bbl@add@sanitize{\@makeother#1}}%
772 \ifx\nfss@catcodes\@undefined\else % TODO - same for above
773 \begingroup
774 \catcode`#1\active
775 \nfss@catcodes
776 \ifnum\catcode`#1=\active
777 \endgroup
778 \bbl@add\nfss@catcodes{\@makeother#1}%
779 \else
780 \endgroup
781 \fi
782 \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.

```

783 \def\bbl@remove@special#1{%
784 \begingroup
785 \def\x##1##2{\ifnum`#1=``##2\noexpand\@empty
786 \else\noexpand##1\noexpand##2\fi}%
787 \def\do{\x\do}%
788 \def\@makeother{\x\@makeother}%
789 \edef\x{\endgroup
790 \def\noexpand\dospecials{\dospecials}%
791 \expandafter\ifx\csgname @sanitize\endcsname\relax\else
792 \def\noexpand\@sanitize{\@sanitize}%
793 \fi}%
794 \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).

```

795 \def\bbl@active@def#1#2#3#4{%
796 \@namedef{#3#1}{%
797 \expandafter\ifx\csgname#2@sh@#1\endcsname\relax

```

```

798 \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
799 \else
800 \bbl@afterfi\csname#2@sh@#1@endcsname
801 \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.

```

802 \long\@namedef{#3@arg#1}##1{%
803 \expandafter\ifx\csname#2@sh@#1@string##1@endcsname\relax
804 \bbl@afterelse\csname#4#1@endcsname##1%
805 \else
806 \bbl@afterfi\csname#2@sh@#1@string##1@endcsname
807 \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.

```

808 \def\initiate@active@char#1{%
809 \bbl@ifunset{active@char\string#1}%
810 {\bbl@withactive
811 {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
812 {}}

```

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`).

```

813 \def\@initiate@active@char#1#2#3{%
814 \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
815 \ifx#1\@undefined
816 \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
817 \else
818 \bbl@csarg\let{oridef@#2}#1%
819 \bbl@csarg\edef{oridef@#2}{%
820 \let\noexpand#1%
821 \expandafter\noexpand\csname bbl@oridef@#2@endcsname}%
822 \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*").

```

823 \ifx#1#3\relax
824 \expandafter\let\csname normal@char#2@endcsname#3%
825 \else
826 \bbl@info{Making #2 an active character}%
827 \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
828 \@namedef{normal@char#2}{%
829 \textormath{#3}{\csname bbl@oridef@#2@endcsname}}%
830 \else
831 \@namedef{normal@char#2}{#3}%
832 \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).

```

833 \bbl@restoreactive{#2}%
834 \AtBeginDocument{%
835   \catcode`#2\active
836   \if@filesw
837     \immediate\write\@mainaux{\catcode`\string#2\active}%
838   \fi}%
839 \expandafter\bbl@add@special\csname#2\endcsname
840 \catcode`#2\active
841 \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⟩`).

```

842 \let\bbl@tempa\@firstoftwo
843 \if\string^#2%
844   \def\bbl@tempa{\noexpand\textormath}%
845 \else
846   \ifx\bbl@mathnormal\@undefined\else
847     \let\bbl@tempa\bbl@mathnormal
848   \fi
849 \fi
850 \expandafter\edef\csname active@char#2\endcsname{%
851   \bbl@tempa
852   {\noexpand\if@safe@actives
853     \noexpand\expandafter
854     \expandafter\noexpand\csname normal@char#2\endcsname
855     \noexpand\else
856       \noexpand\expandafter
857       \expandafter\noexpand\csname bbl@doactive#2\endcsname
858     \noexpand\fi}%
859   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
860 \bbl@csarg\edef{doactive#2}{%
861   \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!).

```

862 \bbl@csarg\edef{active@#2}{%
863   \noexpand\active@prefix\noexpand#1%
864   \expandafter\noexpand\csname active@char#2\endcsname}%
865 \bbl@csarg\edef{normal@#2}{%
866   \noexpand\active@prefix\noexpand#1%
867   \expandafter\noexpand\csname normal@char#2\endcsname}%
868 \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.

```

869 \bbl@active@def#2\user@group{user@active}{language@active}%
870 \bbl@active@def#2\language@group{language@active}{system@active}%
871 \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 T<sub>E</sub>X would see \protect'\protect'. To prevent this from happening a couple of shorthand needs to be defined at user level.

```
872 \expandafter\edef\csname\user@group @sh@#2@@\endcsname
873   {\expandafter\noexpand\csname normal@char#2\endcsname}%
874 \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
875   {\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.

```
876 \if\string'#2%
877   \let\prim@s\bbl@prim@s
878   \let\active@math@prime#1%
879 \fi
880 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}
```

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

```
881 <<{*More package options}>> ≡
882 \DeclareOption{math=active}{}
883 \DeclareOption{math=normal}{{\def\bbl@mathnormal{\noexpand\textormath}}}
884 <</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.

```
885 \ifpackagewith{babel}{KeepShorthandsActive}%
886   {\let\bbl@restoreactive\@gobble}%
887   {\def\bbl@restoreactive#1{%
888     \bbl@exp{%
889       \\\AfterBabelLanguage\\CurrentOption
890       {\catcode`#1=\the\catcode`#1\relax}%
891       \\\AtEndOfPackage
892       {\catcode`#1=\the\catcode`#1\relax}}}%
893   \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.

```
894 \def\bbl@sh@select#1#2{%
895   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
896     \bbl@afterelse\bbl@scndcs
897   \else
898     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
899   \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.

```

900 \begingroup
901 \bbl@ifunset{ifincsname}%
902   {\gdef\active@prefix#1{%
903     \ifx\protect\@typeset@protect
904     \else
905       \ifx\protect\@unexpandable@protect
906       \noexpand#1%
907       \else
908       \protect#1%
909       \fi
910       \expandafter\@gobble
911     \fi}}
912   {\gdef\active@prefix#1{%
913     \ifincsname
914     \string#1%
915     \expandafter\@gobble
916     \else
917     \ifx\protect\@typeset@protect
918     \else
919       \ifx\protect\@unexpandable@protect
920       \noexpand#1%
921       \else
922       \protect#1%
923       \fi
924       \expandafter\expandafter\expandafter\@gobble
925     \fi
926     \fi}}
927 \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⟩.

```

928 \newif\if@safe@actives
929 \@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.

```

930 \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.

```

931 \def\bbl@activate#1{%
932   \bbl@withactive{\expandafter\let\expandafter}#1%
933   \csname bbl@active@\string#1\endcsname}
934 \def\bbl@deactivate#1{%
935   \bbl@withactive{\expandafter\let\expandafter}#1%
936   \csname bbl@normal@\string#1\endcsname}

```

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

```

937 \def\bbl@firstcs#1#2{\csname#1\endcsname}
938 \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.

```

939 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
940 \def\@decl@short#1#2#3\@nil#4{%
941   \def\bbl@tempa{#3}%
942   \ifx\bbl@tempa\@empty
943     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
944     \bbl@ifunset{#1@sh@\string#2@}{}%
945     {\def\bbl@tempa{#4}%
946       \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
947       \else
948         \bbl@info
949           {Redefining #1 shorthand \string#2\\
950            in language \CurrentOption}%
951       \fi}%
952     \@namedef{#1@sh@\string#2@}{#4}%
953   \else
954     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
955     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
956     {\def\bbl@tempa{#4}%
957       \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
958       \else
959         \bbl@info
960           {Redefining #1 shorthand \string#2\string#3\\
961            in language \CurrentOption}%
962       \fi}%
963     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
964   \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.

```

965 \def\textormath{%
966   \ifmmode
967     \expandafter\@secondoftwo
968   \else
969     \expandafter\@firstoftwo
970   \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’.

```

971 \def\user@group{user}
972 \def\language@group{english}
973 \def\system@group{system}

```

`\useshorthands` This is the user level command to tell  $\TeX$  that user level shorthands will be used in the document. It takes one argument, the character that starts a shorthand. First note that this is user level, and then initialize and activate the character for use as a shorthand character (ie, it’s active in the preamble). Languages can deactivate shorthands, so a starred version is also provided which activates them always after the language has been switched.

```

974 \def\useshorthands{%
975   \ifstar\bbl@usesh@s{\bbl@usesh@x{}}
976 \def\bbl@usesh@s#1{%
977   \bbl@usesh@x
978   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
979   {#1}}
980 \def\bbl@usesh@x#1#2{%
981   \bbl@ifshorthand{#2}%
982   {\def\user@group{user}%
983     \initiate@active@char{#2}%
984     #1%
985     \bbl@activate{#2}}%
986   {\bbl@error
987     {Cannot declare a shorthand turned off (\string#2)}
988     {Sorry, but you cannot use shorthands which have been\%
989       turned off in the package options}}}

```

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

```

990 \def\user@language@group{user@\language@group}
991 \def\bbl@set@user@generic#1#2{%
992   \bbl@ifunset{user@generic@active#1}%
993   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
994     \bbl@active@def#1\user@group{user@generic@active}{language@active}%
995     \expandafter\edef\csname#2@sh@#1@@\endcsname{%
996       \expandafter\noexpand\csname normal@char#1\endcsname}%
997     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
998       \expandafter\noexpand\csname user@active#1\endcsname}}%
999   \@empty}
1000 \newcommand\defineshorthand[3][user]{%
1001   \edef\bbl@tempa{\zap@space#1 \@empty}%
1002   \bbl@for\bbl@tempb\bbl@tempa{%
1003     \if*\expandafter\@car\bbl@tempb\@nil
1004       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1005       \@expandtwoargs
1006       \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1007     \fi
1008     \declare@shorthand{\bbl@tempb}{#2}{#3}}}

```

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

```

1009 \def\languageshorthands#1{\def\language@group{#1}}

```

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

```

1010 \def\aliasshorthand#1#2{%
1011   \bbl@ifshorthand{#2}%
1012   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1013     \ifx\document\@notprerr
1014       \@notshorthand{#2}%
1015     \else
1016       \initiate@active@char{#2}%

```

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

littest to \active@char".

```

1017      \expandafter\let\csname active@char\string#2\expandafter\endcsname
1018      \csname active@char\string#1\endcsname
1019      \expandafter\let\csname normal@char\string#2\expandafter\endcsname
1020      \csname normal@char\string#1\endcsname
1021      \bbl@activate{#2}%
1022      \fi
1023      \fi}%
1024      {\bbl@error
1025      {Cannot declare a shorthand turned off (\string#2)}
1026      {Sorry, but you cannot use shorthands which have been\\%
1027      turned off in the package options}}}
```

\@notshorthand

```

1028 \def\@notshorthand#1{%
1029   \bbl@error{%
1030     The character '\string #1' should be made a shorthand character;\\%
1031     add the command \string\useshorthands\string{#1\string} to
1032     the preamble.\\%
1033     I will ignore your instruction}%
1034   {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.

```

1035 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1036 \DeclareRobustCommand*\shorthandoff{%
1037   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1038 \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.

```

1039 \def\bbl@switch@sh#1#2{%
1040   \ifx#2\@nnil\else
1041     \bbl@ifunset{\bbl@active@\string#2}%
1042     {\bbl@error
1043       {I cannot switch '\string#2' on or off--not a shorthand}%
1044       {This character is not a shorthand. Maybe you made\\%
1045       a typing mistake? I will ignore your instruction}}}%
1046     {\ifcase#1%
1047       \catcode'#212\relax
1048       \or
1049       \catcode'#2\active
1050       \or
1051       \csname bbl@oricat@\string#2\endcsname
1052       \csname bbl@oridef@\string#2\endcsname
1053       \fi}%
1054     \bbl@afterfi\bbl@switch@sh#1%
1055   \fi}
```

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

```

1056 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1057 \def\bbl@putsh#1{%
1058   \bbl@ifunset{\bbl@active@\string#1}%
1059   {\bbl@putsh@i#1\@empty\@nnil}%
1060   {\csname bbl@active@\string#1\endcsname}}
1061 \def\bbl@putsh@i#1#2\@nnil{%
1062   \csname\language\sh@\string#1@%
1063     \ifx\@empty#2\else\string#2\fi\endcsname}
1064 \ifx\bbl@opt@shorthands\@nnil\else
1065   \let\bbl@s@initiate@active@char\initiate@active@char
1066   \def\initiate@active@char#1{%
1067     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1068   \let\bbl@s@switch@sh\bbl@switch@sh
1069   \def\bbl@switch@sh#1#2{%
1070     \ifx#2\@nnil\else
1071       \bbl@afterfi
1072       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1073       \fi}
1074   \let\bbl@s@activate\bbl@activate
1075   \def\bbl@activate#1{%
1076     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1077   \let\bbl@s@deactivate\bbl@deactivate
1078   \def\bbl@deactivate#1{%
1079     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1080 \fi

```

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

```

1081 \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**

```

1082 \def\bbl@prim@s{%
1083   \prime\futurelet\@let@token\bbl@pr@m@s}
1084 \def\bbl@if@primes#1#2{%
1085   \ifx#1\@let@token
1086     \expandafter\@firstoftwo
1087   \else\ifx#2\@let@token
1088     \bbl@afterelse\expandafter\@firstoftwo
1089   \else
1090     \bbl@afterfi\expandafter\@secondoftwo
1091   \fi\fi}
1092 \begingroup
1093   \catcode`\^=7 \catcode`\*= \active \lccode`\^=\^
1094   \catcode`\'=12 \catcode`\`= \active \lccode`\`= `
1095   \lowercase{%
1096     \gdef\bbl@pr@m@s{%
1097       \bbl@if@primes" "%
1098       \pr@@@s
1099       {\bbl@if@primes*\^ \pr@@@t\egroup}}
1100 \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).

```
1101 \initiate@active@char{~}
1102 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1103 \bbl@activate{~}
```

\OT1dqpos 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.

```
1104 \expandafter\def\csname OT1dqpos\endcsname{127}
1105 \expandafter\def\csname T1dqpos\endcsname{4}
```

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

```
1106 \ifx\f@encoding\@undefined
1107   \def\f@encoding{OT1}
1108 \fi
```

## 9.5 Language attributes

Language attributes provide a means to give the user control over which features of the language definition files he wants to enable.

\languageattribute The macro \languageattribute checks whether its arguments are valid and then activates the selected language attribute. First check whether the language is known, and then process each attribute in the list.

```
1109 \bbl@trace{Language attributes}
1110 \newcommand\languageattribute[2]{%
1111   \def\bbl@tempc{#1}%
1112   \bbl@fixname\bbl@tempc
1113   \bbl@iflanguage\bbl@tempc{%
1114     \bbl@vforeach{#2}{%
```

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

```
1115     \ifx\bbl@known@attrs\@undefined
1116       \in@false
1117     \else
1118       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
1119     \fi
1120     \ifin@
1121       \bbl@warning{%
1122         You have more than once selected the attribute '##1'\%
1123         for language #1. Reported}%
1124     \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.

```
1125       \bbl@exp{%
1126         \\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
1127       \edef\bbl@tempa{\bbl@tempc-##1}%
1128       \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1129       {\csname\bbl@tempc @attr##1\endcsname}%
1130       {\@attrerr{\bbl@tempc}{##1}}%
1131     \fi}}
```

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

```
1132 \@onlypreamble\languageattribute
```

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

```
1133 \newcommand*{\@attrerr}[2]{%
1134   \bbl@error
1135   {The attribute #2 is unknown for language #1.}%
1136   {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}`.

```
1137 \def\bbl@declare@ttribute#1#2#3{%
1138   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1139   \ifin@
1140     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1141   \fi
1142   \bbl@add@list\bbl@attributes{#1-#2}%
1143   \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.

```
1144 \def\bbl@ifattributeset#1#2#3#4{%
  First we need to find out if any attributes were set; if not we're done.
```

```
1145   \ifx\bbl@known@attribs\undefined
1146     \in@false
1147   \else
```

The we need to check the list of known attributes.

```
1148   \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1149   \fi
```

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

```
1150   \ifin@
1151     \bbl@afterelse#3%
1152   \else
1153     \bbl@afterfi#4%
1154   \fi
1155 }
```

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

```
1156 \def\bbl@ifknown@ttrib#1#2{%
```

We first assume the attribute is unknown.

```
1157   \let\bbl@tempa\@secondoftwo
```



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

```
1158 \bbl@loopx\bbl@tempb{#2}{%
1159   \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{, #1,}%
1160   \ifin@
```

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

```
1161   \let\bbl@tempa\@firstoftwo
1162   \else
1163   \fi}%
```

Finally we execute `\bbl@tempa`.

```
1164 \bbl@tempa
1165 }
```

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

```
1166 \def\bbl@clear@ttribs{%
1167   \ifx\bbl@attributes\@undefined\else
1168     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1169       \expandafter\bbl@clear@ttrib\bbl@tempa.
1170     }%
1171     \let\bbl@attributes\@undefined
1172   \fi}
1173 \def\bbl@clear@ttrib#1-#2.{%
1174   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1175 \AtBeginDocument{\bbl@clear@ttribs}
```

## 9.6 Support for saving macro definitions

To save the meaning of control sequences using `\babel@save`, we use temporary control sequences. To save hash table entries for these control sequences, we don't use the name of the control sequence to be saved to construct the temporary name. Instead we simply use the value of a counter, which is reset to zero each time we begin to save new values. This works well because we release the saved meanings before we begin to save a new set of control sequence meanings (see `\selectlanguage` and `\originalTeX`). Note undefined macros are not undefined any more when saved – they are `\relax`'ed.

`\babel@savecnt` The initialization of a new save cycle: reset the counter to zero.

```
\babel@beginsave 1176 \bbl@trace{Macros for saving definitions}
1177 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1178 \newcount\babel@savecnt
1179 \babel@beginsave
```

`\babel@save` The macro `\babel@save<csname>` saves the current meaning of the control sequence `<csname>` to `\originalTeX`<sup>29</sup>. To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to `\originalTeX` and the counter is incremented. The macro `\babel@savevariable<variable>` saves the value of the variable. `<variable>` can be anything allowed after the `\the` primitive.

```
1180 \def\babel@save#1{%
1181   \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
1182   \toks@\expandafter{\originalTeX\let#1=}
1183   \bbl@exp{%
1184     \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
```

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

```

1185 \advance\babel@savecnt\@ne}
1186 \def\babel@savevariable#1{%
1187 \toks@\expandafter\originalTeX #1=%}
1188 \bbl@exp{\def\\originalTeX{\the\toks@\the#1\relax}}

```

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

```

1189 \def\bbl@frenchspacing{%
1190 \ifnum\the\sfcodes\@m
1191 \let\bbl@nonfrenchspacing\relax
1192 \else
1193 \frenchspacing
1194 \let\bbl@nonfrenchspacing\nonfrenchspacing
1195 \fi}
1196 \let\bbl@nonfrenchspacing\nonfrenchspacing

```

## 9.7 Short tags

`\babeltags` This macro is straightforward. After zapping spaces, we loop over the list and define the  
macros `\text{<tag>}` and `\<tag>`. Definitions are first expanded so that they don't contain  
`\csname` but the actual macro.

```

1197 \bbl@trace{Short tags}
1198 \def\babeltags#1{%
1199 \edef\bbl@tempa{\zap@space#1 \@empty}%
1200 \def\bbl@tempb##1=##2\@{
1201 \edef\bbl@tempc{%
1202 \noexpand\newcommand
1203 \expandafter\noexpand\csname ##1\endcsname{%
1204 \noexpand\protect
1205 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1206 \noexpand\newcommand
1207 \expandafter\noexpand\csname text##1\endcsname{%
1208 \noexpand\foreignlanguage{##2}}}
1209 \bbl@tempc}%
1210 \bbl@for\bbl@tempa\bbl@tempa{%
1211 \expandafter\bbl@tempb\bbl@tempa\@{}}

```

## 9.8 Hyphens

`\babelhyphenation` This macro saves hyphenation exceptions. Two macros are used to store them:  
`\bbl@hyphenation@` for the global ones and `\bbl@hyphenation<lang>` for language ones.  
See `\bbl@patterns` above for further details. We make sure there is a space between  
words when multiple commands are used.

```

1212 \bbl@trace{Hyphens}
1213 \@onlypreamble\babelhyphenation
1214 \AtEndOfPackage{%
1215 \newcommand\babelhyphenation[2][\@empty]{%
1216 \ifx\bbl@hyphenation@\relax
1217 \let\bbl@hyphenation@\@empty
1218 \fi
1219 \ifx\bbl@hyphlist\@empty\else
1220 \bbl@warning{%
1221 You must not intermingle \string\selectlanguage\space and\\%
1222 \string\babelhyphenation\space or some exceptions will not\\%
1223 be taken into account. Reported}%

```

```

1224 \fi
1225 \ifx\@empty#1%
1226 \protected@edef\bbl@hyphenation@{\bbl@hyphenation@space#2}%
1227 \else
1228 \bbl@vforeach{#1}{%
1229 \def\bbl@tempa{##1}%
1230 \bbl@fixname\bbl@tempa
1231 \bbl@iflanguage\bbl@tempa{%
1232 \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1233 \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1234 \@empty
1235 {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1236 #2}}}%
1237 \fi}}

```

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

```

1238 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1239 \def\bbl@t@one{T1}
1240 \def\allowhyphens{\ifx\cf@encoding\bbl@t@one\else\bbl@allowhyphens\fi}

```

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

```

1241 \newcommand\babellnullhyphen{\char\hyphenchar\font}
1242 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1243 \def\bbl@hyphen{%
1244 \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
1245 \def\bbl@hyphen@i#1#2{%
1246 \bbl@ifunset{bbl@hy@#1#2\@empty}%
1247 {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{-}{#2}}}%
1248 {\csname bbl@hy@#1#2\@empty\endcsname}}

```

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

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.

```

1249 \def\bbl@usehyphen#1{%
1250 \leavevmode
1251 \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1252 \nobreak\hskip\z@skip}
1253 \def\bbl@@usehyphen#1{%
1254 \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1255 \def\bbl@hyphenchar{%
1256 \ifnum\hyphenchar\font=\m@ne
1257 \babellnullhyphen
1258 \else
1259 \char\hyphenchar\font
1260 \fi}

```

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

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

```

1261 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1262 \def\bbl@hy@@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1263 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1264 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1265 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1266 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1267 \def\bbl@hy@repeat{%
1268   \bbl@usehyphen{%
1269     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1270 \def\bbl@hy@@repeat{%
1271   \bbl@usehyphen{%
1272     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1273 \def\bbl@hy@empty{\hskip\z@skip}
1274 \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.

```

1275 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#1}\bbl@allowhyphens}

```

## 9.9 Multiencoding strings

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

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

```

1276 \bbl@trace{Multiencoding strings}
1277 \def\bbl@tglobal#1{\global\let#1#1}
1278 \def\bbl@recatcode#1{%
1279   \@tempcnta="7F
1280   \def\bbl@tempa{%
1281     \ifnum\@tempcnta>"FF\else
1282       \catcode\@tempcnta=#1\relax
1283       \advance\@tempcnta\@ne
1284       \expandafter\bbl@tempa
1285     \fi}%
1286   \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).

```

1287 \@ifpackagewith{babel}{nocase}%
1288   {\let\bbl@patchuclc\relax}%
1289   {\def\bbl@patchuclc{%

```

```

1290 \global\let\bbl@patchuclc\relax
1291 \g@addto@macro\uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1292 \gdef\bbl@uclc##1{%
1293   \let\bbl@encoded\bbl@encoded@uclc
1294   \bbl@ifunset{\language @bbl@uclc}% and resumes it
1295   {##1}%
1296   {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1297     \csname\language @bbl@uclc\endcsname}%
1298     {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
1299   \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
1300   \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}
1301 <<(*More package options)>> ≡
1302 \DeclareOption{nocase}{}
1303 <</More package options>>

```

The following package options control the behavior of \SetString.

```

1304 <<(*More package options)>> ≡
1305 \let\bbl@opt@strings\@nnil % accept strings=value
1306 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1307 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1308 \def\BabelStringsDefault{generic}
1309 <</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.

```

1310 \@onlypreamble\StartBabelCommands
1311 \def\StartBabelCommands{%
1312   \begingroup
1313   \bbl@recatcode{11}%
1314   <<Macros local to BabelCommands>>
1315   \def\bbl@provstring##1##2{%
1316     \providecommand##1{##2}%
1317     \bbl@tglobal##1}%
1318   \global\let\bbl@scafter\@empty
1319   \let\StartBabelCommands\bbl@startcmds
1320   \ifx\BabelLanguages\relax
1321     \let\BabelLanguages\CurrentOption
1322   \fi
1323   \begingroup
1324   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1325   \StartBabelCommands}
1326 \def\bbl@startcmds{%
1327   \ifx\bbl@screset\@nnil\else
1328     \bbl@usehooks{stopcommands}{}%
1329   \fi
1330   \endgroup
1331   \begingroup
1332   \@ifstar
1333   {\ifx\bbl@opt@strings\@nnil
1334     \let\bbl@opt@strings\BabelStringsDefault
1335   \fi
1336   \bbl@startcmds@i}%
1337   \bbl@startcmds@i}
1338 \def\bbl@startcmds@i#1#2{%
1339   \edef\bbl@L{\zap@space#1 \@empty}%
1340   \edef\bbl@G{\zap@space#2 \@empty}%

```

```

1341 \bbl@startcmds@ii}
1342 \let\bbl@startcommands\StartBabelCommands

```

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

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

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

```

1343 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1344 \let\SetString\@gobbletwo
1345 \let\bbl@stringdef\@gobbletwo
1346 \let\AfterBabelCommands\@gobble
1347 \ifx\@empty#1%
1348 \def\bbl@sc@label{generic}%
1349 \def\bbl@encstring##1##2{%
1350 \ProvideTextCommandDefault##1{##2}%
1351 \bbl@tglobal##1%
1352 \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%
1353 \let\bbl@sctest\in@true
1354 \else
1355 \let\bbl@sc@charset\space % <- zapped below
1356 \let\bbl@sc@fontenc\space % <- " "
1357 \def\bbl@tempa##1=##2\@nil{%
1358 \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1359 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1360 \def\bbl@tempa##1 ##2{% space -> comma
1361 ##1%
1362 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1363 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1364 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1365 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1366 \def\bbl@encstring##1##2{%
1367 \bbl@foreach\bbl@sc@fontenc{%
1368 \bbl@ifunset{T@####1}%
1369 }%
1370 {\ProvideTextCommand##1{####1}{##2}%
1371 \bbl@tglobal##1%
1372 \expandafter
1373 \bbl@tglobal\csname####1\string##1\endcsname}}}%
1374 \def\bbl@sctest{%
1375 \bbl@xin@{,\bbl@opt@strings,},{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1376 \fi
1377 \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
1378 \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
1379 \let\AfterBabelCommands\bbl@aftercmds
1380 \let\SetString\bbl@setstring
1381 \let\bbl@stringdef\bbl@encstring
1382 \else % ie, strings=value
1383 \bbl@sctest
1384 \ifin@
1385 \let\AfterBabelCommands\bbl@aftercmds
1386 \let\SetString\bbl@setstring
1387 \let\bbl@stringdef\bbl@provstring

```

```

1388 \fi\fi\fi
1389 \bbl@scswitch
1390 \ifx\bbl@G\@empty
1391 \def\SetString##1##2{%
1392     \bbl@error{Missing group for string \string##1}%
1393     {You must assign strings to some category, typically\\%
1394     captions or extras, but you set none}}%
1395 \fi
1396 \ifx\@empty#1%
1397     \bbl@usehooks{defaultcommands}{}%
1398 \else
1399     \@expandtwoargs
1400     \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1401 \fi}

```

There are two versions of `\bbl@scswitch`. The first version is used when `ldfs` are read, and it makes sure `\group``\language` 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\language` 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).

```

1402 \def\bbl@forlang#1#2{%
1403     \bbl@for#1\bbl@L{%
1404         \bbl@xin@{, #1, }{, \BabelLanguages, }%
1405         \ifin@#2\relax\fi}}
1406 \def\bbl@scswitch{%
1407     \bbl@forlang\bbl@tempa{%
1408         \ifx\bbl@G\@empty\else
1409             \ifx\SetString\@gobbletwo\else
1410                 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1411                 \bbl@xin@{, \bbl@GL, }{, \bbl@screset, }%
1412                 \ifin@\else
1413                     \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1414                     \xdef\bbl@screset{\bbl@screset, \bbl@GL}%
1415                 \fi
1416             \fi
1417         \fi}}
1418 \AtEndOfPackage{%
1419     \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
1420     \let\bbl@scswitch\relax}
1421 \@onlypreamble\EndBabelCommands
1422 \def\EndBabelCommands{%
1423     \bbl@usehooks{stopcommands}{}%
1424     \endgroup
1425     \endgroup
1426     \bbl@scafter}
1427 \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.

```

1428 \def\bbl@setstring#1#2{%
1429   \bbl@forlang\bbl@tempa{%
1430     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1431     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1432     {\global\expandafter % TODO - con \bbl@exp ?
1433       \bbl@add\csname\bbl@G\bbl@tempa\expandafter\endcsname\expandafter
1434         {\expandafter\bbl@scset\expandafter#1\csname\bbl@LC\endcsname}}%
1435     }%
1436   \def\BabelString{#2}%
1437   \bbl@usehooks{stringprocess}{}%
1438   \expandafter\bbl@stringdef
1439     \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`.

```

1440 \ifx\bbl@opt@strings\relax
1441   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1442   \bbl@patchuclc
1443   \let\bbl@encoded\relax
1444   \def\bbl@encoded@uclc#1{%
1445     \@inmathwarn#1%
1446     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1447       \expandafter\ifx\csname ?\string#1\endcsname\relax
1448         \TextSymbolUnavailable#1%
1449       \else
1450         \csname ?\string#1\endcsname
1451       \fi
1452     \else
1453       \csname\cf@encoding\string#1\endcsname
1454     \fi}
1455 \else
1456   \def\bbl@scset#1#2{\def#1{#2}}
1457 \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.

```

1458 << *Macros local to BabelCommands >> ≡
1459 \def\SetStringLoop##1##2{%
1460   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1461   \count@\z@
1462   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1463     \advance\count@\@ne
1464     \toks@\expandafter{\bbl@tempa}%
1465     \bbl@exp{%
1466       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1467       \count@=\the\count@\relax}}}%
1468 << /Macros local to BabelCommands >>

```

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

```

1469 \def\bbl@aftercmds#1{%
1470   \toks@\expandafter{\bbl@scafter#1}%
1471   \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.

```

1472 <<*Macros local to BabelCommands>> ≡
1473 \newcommand\SetCase[3][]{%
1474   \bbl@patchuclc
1475   \bbl@forlang\bbl@tempa{%
1476     \expandafter\bbl@encstring
1477     \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
1478     \expandafter\bbl@encstring
1479     \csname\bbl@tempa @bbl@uc\endcsname{##2}%
1480     \expandafter\bbl@encstring
1481     \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
1482 <</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.

```

1483 <<*Macros local to BabelCommands>> ≡
1484 \newcommand\SetHyphenMap[1]{%
1485   \bbl@forlang\bbl@tempa{%
1486     \expandafter\bbl@stringdef
1487     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1488 <</Macros local to BabelCommands>>

```

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

```

1489 \newcommand\BabelLower[2]{% one to one.
1490   \ifnum\lccode#1=#2\else
1491     \babel@savevariable{\lccode#1}%
1492     \lccode#1=#2\relax
1493   \fi}
1494 \newcommand\BabelLowerMM[4]{% many-to-many
1495   \@tempcnta=#1\relax
1496   \@tempcntb=#4\relax
1497   \def\bbl@tempa{%
1498     \ifnum\@tempcnta>#2\else
1499       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1500       \advance\@tempcnta#3\relax
1501       \advance\@tempcntb#3\relax
1502       \expandafter\bbl@tempa
1503     \fi}%
1504   \bbl@tempa}
1505 \newcommand\BabelLowerMO[4]{% many-to-one
1506   \@tempcnta=#1\relax
1507   \def\bbl@tempa{%
1508     \ifnum\@tempcnta>#2\else
1509       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1510       \advance\@tempcnta#3
1511       \expandafter\bbl@tempa
1512     \fi}%
1513   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1514 <<*More package options>> ≡
1515 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1516 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1517 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1518 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}

```

```

1519 \DeclareOption{hyphenmap=other*}{\chardef\bb1@opt@hyphenmap4\relax}
1520 <</More package options>>

```

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

```

1521 \AtEndOfPackage{%
1522   \ifx\bb1@opt@hyphenmap\undefined
1523     \bb1@xin@{,}{\bb1@language@opts}%
1524     \chardef\bb1@opt@hyphenmap\ifin@4\else\@ne\fi
1525   \fi}

```

## 9.10 Macros common to a number of languages

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

```

1526 \bb1@trace{Macros related to glyphs}
1527 \def\set@low@box#1{\setbox\tw\hbox{,}\setbox\z\hbox{#1}%
1528   \dimen\z\ht\z@ \advance\dimen\z@ -\ht\tw@%
1529   \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.

```

1530 \def\save@sf@q#1{\leavevmode
1531   \begingroup
1532   \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
1533   \endgroup}

```

## 9.11 Making glyphs available

This section makes a number of glyphs available that either do not exist in the OT1 encoding and have to be ‘faked’, or that are not accessible through `T1enc.def`.

### 9.11.1 Quotation marks

`\quotedblbase` In the T1 encoding the opening double quote at the baseline is available as a separate character, accessible via `\quotedblbase`. In the OT1 encoding it is not available, therefore we make it available by lowering the normal open quote character to the baseline.

```

1534 \ProvideTextCommand{\quotedblbase}{OT1}{%
1535   \save@sf@q{\set@low@box{\textquotedblright\}}%
1536   \box\z@\kern-.04em\bb1@allowhyphens}}

```

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

```

1537 \ProvideTextCommandDefault{\quotedblbase}{%
1538   \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

1539 \ProvideTextCommand{\quotesinglbase}{OT1}{%
1540   \save@sf@q{\set@low@box{\textquoteright\}}%
1541   \box\z@\kern-.04em\bb1@allowhyphens}}

```

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

```

1542 \ProvideTextCommandDefault{\quotesinglbase}{%
1543   \UseTextSymbol{OT1}{\quotesinglbase}}

```

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

```
\guillemotright 1544 \ProvideTextCommand{\guillemotleft}{OT1}{%
1545   \ifmmode
1546     \ll
1547   \else
1548     \save@sf@q{\nobreak
1549       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
1550   \fi}
1551 \ProvideTextCommand{\guillemotright}{OT1}{%
1552   \ifmmode
1553     \gg
1554   \else
1555     \save@sf@q{\nobreak
1556       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
1557   \fi}
```

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

```
1558 \ProvideTextCommandDefault{\guillemotleft}{%
1559   \UseTextSymbol{OT1}{\guillemotleft}}
1560 \ProvideTextCommandDefault{\guillemotright}{%
1561   \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
\guilsinglright 1562 \ProvideTextCommand{\guilsinglleft}{OT1}{%
1563   \ifmmode
1564     <%
1565   \else
1566     \save@sf@q{\nobreak
1567       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
1568   \fi}
1569 \ProvideTextCommand{\guilsinglright}{OT1}{%
1570   \ifmmode
1571     >%
1572   \else
1573     \save@sf@q{\nobreak
1574       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
1575   \fi}
```

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

```
1576 \ProvideTextCommandDefault{\guilsinglleft}{%
1577   \UseTextSymbol{OT1}{\guilsinglleft}}
1578 \ProvideTextCommandDefault{\guilsinglright}{%
1579   \UseTextSymbol{OT1}{\guilsinglright}}
```

### 9.11.2 Letters

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

```
1580 \DeclareTextCommand{\ij}{OT1}{%
1581   i\kern-0.02em\bbl@allowhyphens j}
1582 \DeclareTextCommand{\IJ}{OT1}{%
1583   I\kern-0.02em\bbl@allowhyphens J}
1584 \DeclareTextCommand{\ij}{T1}{\char188}
1585 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
1586 \ProvideTextCommandDefault{\ij}{%
1587   \UseTextSymbol{OT1}{\ij}}
1588 \ProvideTextCommandDefault{\IJ}{%
1589   \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).

```
1590 \def\crrtic@{\hrule height0.1ex width0.3em}
1591 \def\crttic@{\hrule height0.1ex width0.33em}
1592 \def\ddj@{%
1593   \setbox0\hbox{d}\dimen@=\ht0
1594   \advance\dimen@1ex
1595   \dimen@.45\dimen@
1596   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1597   \advance\dimen@ii.5ex
1598   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
1599 \def\DDJ@{%
1600   \setbox0\hbox{D}\dimen@=.55\ht0
1601   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1602   \advance\dimen@ii.15ex % correction for the dash position
1603   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
1604   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
1605   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
1606 %
1607 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
1608 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
1609 \ProvideTextCommandDefault{\dj}{%
1610   \UseTextSymbol{OT1}{\dj}}
1611 \ProvideTextCommandDefault{\DJ}{%
1612   \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.

```
1613 \DeclareTextCommand{\SS}{OT1}{SS}
1614 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

### 9.11.3 Shorthands for quotation marks

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

`\glq` The ‘german’ single quotes.

```
\grq 1615 \ProvideTextCommandDefault{\glq}{%
1616   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```

1617 \ProvideTextCommand{\grq}{T1}{%
1618   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
1619 \ProvideTextCommand{\grq}{TU}{%
1620   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
1621 \ProvideTextCommand{\grq}{OT1}{%
1622   \save@sf@q{\kern-.0125em
1623     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
1624     \kern.07em\relax}}
1625 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}

```

\glqq The ‘german’ double quotes.

```

\grqq 1626 \ProvideTextCommandDefault{\glqq}{%
1627   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

1628 \ProvideTextCommand{\grqq}{T1}{%
1629   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1630 \ProvideTextCommand{\grqq}{TU}{%
1631   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1632 \ProvideTextCommand{\grqq}{OT1}{%
1633   \save@sf@q{\kern-.07em
1634     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
1635     \kern.07em\relax}}
1636 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}

```

\flq The ‘french’ single guillemets.

```

\frq 1637 \ProvideTextCommandDefault{\flq}{%
1638   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
1639 \ProvideTextCommandDefault{\frq}{%
1640   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

\flqq The ‘french’ double guillemets.

```

\frqq 1641 \ProvideTextCommandDefault{\flqq}{%
1642   \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
1643 \ProvideTextCommandDefault{\frqq}{%
1644   \textormath{\guillemotright}{\mbox{\guillemotright}}}

```

#### 9.11.4 Umlauts and tremas

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

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

```

1645 \def\umlauthigh{%
1646   \def\bb1@umlauta##1{\leavevmode\bggroup%
1647     \expandafter\accent\csname\fontencoding dqpos\endcsname
1648     ##1\bb1@allowhyphens\egroup}%
1649   \let\bb1@umlaute\bb1@umlauta}
1650 \def\umlautlow{%
1651   \def\bb1@umlauta{\protect\lower@umlaut}}
1652 \def\umlautelow{%
1653   \def\bb1@umlaute{\protect\lower@umlaut}}
1654 \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.

```
1655 \expandafter\ifx\csname U@D\endcsname\relax
1656 \csname newdimen\endcsname\U@D
1657 \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.

```
1658 \def\lower@umlaut#1{%
1659   \leavevmode\bggroup
1660     \U@D 1ex%
1661     {\setbox\z@\hbox{%
1662       \expandafter\char\csname\fontencoding dqpos\endcsname}%
1663       \dimen@ -.45ex\advance\dimen@\ht\z@
1664       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
1665     \expandafter\accent\csname\fontencoding dqpos\endcsname
1666     \fontdimen5\font\U@D #1%
1667   \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).

```
1668 \AtBeginDocument{%
1669   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
1670   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
1671   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{~i}}%
1672   \DeclareTextCompositeCommand{\}{OT1}{~i}{\bbl@umlaute{~i}}%
1673   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
1674   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
1675   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
1676   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
1677   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
1678   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
1679   \DeclareTextCompositeCommand{\}{OT1}{U}{\bbl@umlauta{U}}%
1680 }
```

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

```
1681 \ifx\l@english\undefined
1682   \chardef\l@english\z@
1683 \fi
1684 \main@language{english}
```

## 9.12 Layout

**Work in progress.**

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

```

1685 \bbl@trace{Bidi layout}
1686 \providecommand\IfBabelLayout[3]{#3}%
1687 \newcommand\BabelPatchSection[1]{%
1688   \@ifundefined{#1}{}{%
1689     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
1690     \@namedef{#1}{%
1691       \ifstar{\bbl@presec@s{#1}}%
1692       {\@dblarg{\bbl@presec@x{#1}}}}}%
1693 \def\bbl@presec@x#1[#2]#3{%
1694   \bbl@exp{%
1695     \\\select@language@x{\bbl@main@language}%
1696     \\\bbl@cs{sspre@#1}%
1697     \\\bbl@cs{ss@#1}%
1698     [\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
1699     {\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
1700     \\\select@language@x{\languagename}}}%
1701 \def\bbl@presec@s#1#2{%
1702   \bbl@exp{%
1703     \\\select@language@x{\bbl@main@language}%
1704     \\\bbl@cs{sspre@#1}%
1705     \\\bbl@cs{ss@#1}*%
1706     {\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
1707     \\\select@language@x{\languagename}}}%
1708 \IfBabelLayout{sectioning}%
1709   {\BabelPatchSection{part}%
1710    \BabelPatchSection{chapter}%
1711    \BabelPatchSection{section}%
1712    \BabelPatchSection{subsection}%
1713    \BabelPatchSection{subsubsection}%
1714    \BabelPatchSection{paragraph}%
1715    \BabelPatchSection{subparagraph}%
1716    \def\babel@toc#1{%
1717      \select@language@x{\bbl@main@language}}}%
1718 \IfBabelLayout{captions}%
1719   {\BabelPatchSection{caption}}}%

```

### 9.13 Load engine specific macros

```

1720 \bbl@trace{Input engine specific macros}
1721 \ifcase\bbl@engine
1722   \input txtbabel.def
1723 \or
1724   \input luababel.def
1725 \or
1726   \input xebabel.def
1727 \fi

```

### 9.14 Creating 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.

```

1728 \bbl@trace{Creating languages and reading ini files}
1729 \newcommand\babelprovide[2][{}]{%
1730   \let\bbl@savelangname\languagename
1731   \edef\bbl@savlocaleid{\the\localeid}%

```

```

1732 % Set name and locale id
1733 \edef\languagename{#2}%
1734 % \global\@namedef{bbl@lcname@#2}{#2}%
1735 \bbl@id@assign
1736 \let\bbl@KVP@captions\@nil
1737 \let\bbl@KVP@import\@nil
1738 \let\bbl@KVP@main\@nil
1739 \let\bbl@KVP@script\@nil
1740 \let\bbl@KVP@language\@nil
1741 \let\bbl@KVP@hyphenrules\@nil % only for provide@new
1742 \let\bbl@KVP@mapfont\@nil
1743 \let\bbl@KVP@maparabic\@nil
1744 \let\bbl@KVP@mapdigits\@nil
1745 \let\bbl@KVP@intraspace\@nil
1746 \let\bbl@KVP@intrapenalty\@nil
1747 \let\bbl@KVP@onchar\@nil
1748 \let\bbl@KVP@alph\@nil
1749 \let\bbl@KVP@Alph\@nil
1750 \let\bbl@KVP@info\@nil % Ignored with import? Or error/warning?
1751 \bbl@forkv{#1}{% TODO - error handling
1752   \in@{/{}}{##1}%
1753   \ifin@
1754     \bbl@renewinikey##1\@{##2}%
1755   \else
1756     \bbl@csarg\def{KVP@##1}{##2}%
1757   \fi}%
1758 % == import, captions ==
1759 \ifx\bbl@KVP@import\@nil\else
1760   \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
1761   {\beginingroup
1762     \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
1763     \InputIfFileExists{babel-#2.tex}{}%
1764     \endgroup}%
1765   {%
1766   \fi
1767   \ifx\bbl@KVP@captions\@nil
1768     \let\bbl@KVP@captions\bbl@KVP@import
1769   \fi
1770   % Load ini
1771   \bbl@ifunset{date#2}%
1772   {\bbl@provide@new{#2}%
1773   {\bbl@ifblank{#1}%
1774     {\bbl@error
1775       {If you want to modify `#2' you must tell how in\\
1776       the optional argument. See the manual for the\\
1777       available options.}%
1778       {Use this macro as documented}}%
1779     {\bbl@provide@renew{#2}}}%
1780   % Post tasks
1781   \bbl@exp{\bbl@babelensure[exclude=\\today]{#2}}%
1782   \bbl@ifunset{bbl@ensure@\languagename}%
1783   {\bbl@exp{%
1784     \\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
1785       \\foreignlanguage{\languagename}%
1786       {###1}}}%
1787   {%
1788   % At this point all parameters are defined if 'import'. Now we
1789   % execute some code depending on them. But what about if nothing was
1790   % imported? We just load the very basic parameters: ids and a few

```



```

1791 % more.
1792 \bbl@ifunset{bbl@lname@#2}%
1793 {\def\BabelBeforeIni##1##2{%
1794     \begingroup
1795     \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\;=12 %
1796     \let\bbl@ini@captions@aux\@gobbletwo
1797     \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6{%
1798     \bbl@read@ini{##1}{basic data}%
1799     \bbl@exportkey{chrng}{characters.ranges}{}%
1800     \bbl@exportkey{dgnat}{numbers.digits.native}{}%
1801     \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
1802     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
1803     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
1804     \bbl@exportkey{hyoth}{typography.hyphenate.other}{}%
1805     \bbl@exportkey{intsp}{typography.intraspace}{}%
1806     \endinput
1807     \endgroup}%
1808     \boxed, to avoid extra spaces:
1809     {\setbox\z@\hbox{\InputIfFileExists{babel-#2.tex}{}}}%
1810 }%
1811 % -
1812 % == script, language ==
1813 % Override the values from ini or defines them
1814 \ifx\bbl@KVP@script\@nil\else
1815     \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
1816 \fi
1817 \ifx\bbl@KVP@language\@nil\else
1818     \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
1819 \fi
1820 % == onchar ==
1821 \ifx\bbl@KVP@onchar\@nil\else
1822     \bbl@luahyphenate
1823     \directlua{
1824         if Babel.locale_mapped == nil then
1825             Babel.locale_mapped = true
1826             Babel.linebreaking.add_before(Babel.locale_map)
1827             Babel.loc_to_scr = {}
1828             Babel.chr_to_loc = Babel.chr_to_loc or {}
1829         end}%
1830 \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
1831 \ifin@
1832     \ifx\bbl@starthyphens\undefined % Needed if no explicit selection
1833         \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
1834     \fi
1835     \bbl@exp{\bbl@add\bbl@starthyphens
1836         {\bbl@patterns@lua{\language}}}%
1837     % TODO - error/warning if no script
1838     \directlua{
1839         if Babel.script_blocks['\bbl@cl{sbc}'] then
1840             Babel.loc_to_scr[\the\localeid] =
1841             Babel.script_blocks['\bbl@cl{sbc}']
1842             Babel.locale_props[\the\localeid].lc = \the\localeid\space
1843             Babel.locale_props[\the\localeid].lg = \the\nameuse{l@language}\space
1844         end
1845     }%
1846 \fi
1847 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
1848 \ifin@
1849     \bbl@ifunset{bbl@lsys\language}{\bbl@provide@lsys\language}{}%
1850     \bbl@ifunset{bbl@wdir\language}{\bbl@provide@dirs\language}{}%

```

```

1850 \directlua{
1851   if Babel.script_blocks['\bbl@cl{sbc}'] then
1852     Babel.loc_to_scr[\the\localeid] =
1853       Babel.script_blocks['\bbl@cl{sbc}']
1854   end}%
1855 \ifx\bbl@mapselect\undefined
1856   \AtBeginDocument{%
1857     \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
1858     {\selectfont}}%
1859   \def\bbl@mapselect{%
1860     \let\bbl@mapselect\relax
1861     \edef\bbl@prefontid{\fontid\font}}%
1862   \def\bbl@mapdir##1{%
1863     {\def\language##1}%
1864     \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
1865     \bbl@switchfont
1866     \directlua{
1867       Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
1868         ['\bbl@prefontid'] = \fontid\font\space}}%
1869   \fi
1870   \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
1871   \fi
1872   % TODO - catch non-valid values
1873 \fi
1874 % == mapfont ==
1875 % For bidi texts, to switch the font based on direction
1876 \ifx\bbl@KVP@mapfont\@nil\else
1877   \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}%
1878   {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\%
1879     mapfont. Use 'direction'.%
1880     {See the manual for details.}}}%
1881   \bbl@ifunset{\bbl@sys@\language}{\bbl@provide@sys@\language}}%
1882   \bbl@ifunset{\bbl@wdir@\language}{\bbl@provide@dirs@\language}}%
1883 \ifx\bbl@mapselect\undefined
1884   \AtBeginDocument{%
1885     \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
1886     {\selectfont}}%
1887   \def\bbl@mapselect{%
1888     \let\bbl@mapselect\relax
1889     \edef\bbl@prefontid{\fontid\font}}%
1890   \def\bbl@mapdir##1{%
1891     {\def\language##1}%
1892     \let\bbl@ifrestoring\@firstoftwo % avoid font warning
1893     \bbl@switchfont
1894     \directlua{Babel.fontmap
1895       [\the\csname bbl@wdir@##1\endcsname]%
1896       [\bbl@prefontid]=\fontid\font}}%
1897   \fi
1898   \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
1899   \fi
1900 % == intraspace, intrapenalty ==
1901 % For CJK, East Asian, Southeast Asian, if interspace in ini
1902 \ifx\bbl@KVP@intraspace\@nil\else % We can override the ini or set
1903   \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
1904 \fi
1905 \bbl@provide@intraspace
1906 % == hyphenate.other ==
1907 \bbl@ifunset{\bbl@hyoth@\language}}%
1908 {\bbl@csarg\bbl@replace{hyoth@\language}{ }{,}}%

```

```

1909 \bbl@startcommands*{\language\name}{}%
1910 \bbl@csarg\bbl@foreach{hyoth\language\name}{%
1911 \ifcase\bbl@engine
1912 \ifnum##1<257
1913 \SetHyphenMap{\BabelLower{##1}{##1}}%
1914 \fi
1915 \else
1916 \SetHyphenMap{\BabelLower{##1}{##1}}%
1917 \fi}%
1918 \bbl@endcommands}
1919 % == maparabic ==
1920 % Native digits, if provided in ini (TeX level, xe and lua)
1921 \ifcase\bbl@engine\else
1922 \bbl@ifunset{\bbl@dgnat\language\name}{}%
1923 {\expandafter\ifx\csname\bbl@dgnat\language\name\endcsname\@empty\else
1924 \expandafter\expandafter\expandafter
1925 \bbl@setdigits\csname\bbl@dgnat\language\name\endcsname
1926 \ifx\bbl@KVP@maparabic\@nil\else
1927 \ifx\bbl@latinarabic\@undefined
1928 \expandafter\let\expandafter\@arabic
1929 \csname\bbl@counter\language\name\endcsname
1930 \else % ie, if layout=counters, which redefines \@arabic
1931 \expandafter\let\expandafter\bbl@latinarabic
1932 \csname\bbl@counter\language\name\endcsname
1933 \fi
1934 \fi
1935 \fi}%
1936 \fi
1937 % == mapdigits ==
1938 % Native digits (lua level).
1939 \ifodd\bbl@engine
1940 \ifx\bbl@KVP@mapdigits\@nil\else
1941 \bbl@ifunset{\bbl@dgnat\language\name}{}%
1942 {\RequirePackage{luatexbase}%
1943 \bbl@activate@preotf
1944 \directlua{
1945 Babel = Babel or {} %%% -> presets in luababel
1946 Babel.digits_mapped = true
1947 Babel.digits = Babel.digits or {}
1948 Babel.digits[\the\localeid] =
1949 table.pack(string.utfvalue('\bbl@cl{dgnat}'))
1950 if not Babel.numbers then
1951 function Babel.numbers(head)
1952 local LOCALE = luatexbase.registernumber'\bbl@attr@locale'
1953 local GLYPH = node.id'glyph'
1954 local inmath = false
1955 for item in node.traverse(head) do
1956 if not inmath and item.id == GLYPH then
1957 local temp = node.get_attribute(item, LOCALE)
1958 if Babel.digits[temp] then
1959 local chr = item.char
1960 if chr > 47 and chr < 58 then
1961 item.char = Babel.digits[temp][chr-47]
1962 end
1963 end
1964 elseif item.id == node.id'math' then
1965 inmath = (item.subtype == 0)
1966 end
1967 end

```

```

1968             return head
1969         end
1970     end
1971 }}%
1972 \fi
1973 \fi
1974 % == alph, Alph ==
1975 % What if extras<lang> contains a \babel@save\@alph? It won't be
1976 % restored correctly when exiting the language, so we ignore
1977 % this change with the \bbl@alph@saved trick.
1978 \ifx\bbl@KVP@alph\@nil\else
1979     \toks@\expandafter\expandafter\expandafter{%
1980         \csname extras\language\endcsname}%
1981     \bbl@exp{%
1982         \def\<extras\language>{%
1983             \let\\bbl@alph@saved\\@alph
1984             \the\toks@
1985             \let\\@alph\\bbl@alph@saved
1986             \\babel@save\\@alph
1987             \let\\@alph\<bbl@cntr@\bbl@KVP@alph @\language>}}%
1988     \fi
1989 \ifx\bbl@KVP@Alph\@nil\else
1990     \toks@\expandafter\expandafter\expandafter{%
1991         \csname extras\language\endcsname}%
1992     \bbl@exp{%
1993         \def\<extras\language>{%
1994             \let\\bbl@Alph@saved\\@Alph
1995             \the\toks@
1996             \let\\@Alph\\bbl@Alph@saved
1997             \\babel@save\\@Alph
1998             \let\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\language>}}%
1999     \fi
2000 % == require.babel in ini ==
2001 % To load or reload the babel-*.tex, if require.babel in ini
2002 \bbl@ifunset{\bbl@rtex\language}{}%
2003 {\expandafter\ifx\csname\bbl@rtex\language\endcsname\@empty\else
2004     \let\BabelBeforeIni\@gobbles
2005     \chardef\atcatcode=\catcode\@
2006     \catcode\@=11\relax
2007     \InputIfFileExists{babel-\bbl@cs{rtex\language}.tex}{}%
2008     \catcode\@=\atcatcode
2009     \let\atcatcode\relax
2010 } \fi}%
2011 % == main ==
2012 \ifx\bbl@KVP@main\@nil % Restore only if not 'main'
2013     \let\language\bbl@savelangname
2014     \chardef\localeid\bbl@savelocaleid\relax
2015 \fi}

```

A tool to define the macros for native digits from the list provided in the ini file.  
Somewhat convoluted because there are 10 digits, but only 9 arguments in  $\TeX$ .

```

2016 \def\bbl@setdigits#1#2#3#4#5{%
2017     \bbl@exp{%
2018         \def\<\language digits>####1{% ie, \langdigits
2019             \<bbl@digits\language>####1\\@nil}%
2020         \def\<\language counter>####1{% ie, \langcounter
2021             \\expandafter\<bbl@counter@\language>%
2022             \\csname c####1\endcsname}%
2023         \def\<bbl@counter@\language>####1{% ie, \bbl@counter@lang

```

[illegible]

```

2046 \def\bbl@provide@new#1{%
2047   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2048   \@namedef{extras#1}{}%
2049   \@namedef{noextras#1}{}%
2050   \bbl@startcommands*{#1}{captions}%
2051   \ifx\bbl@KVP@captions\@nil %      and also if import, implicit
2052     \def\bbl@tempb##1{%             elt for \bbl@captionslist
2053       \ifx##1\@empty\else
2054         \bbl@exp{%
2055           \\SetString\\##1{%
2056             \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2057           \expandafter\bbl@tempb
2058         \fi}%
2059     \expandafter\bbl@tempb\bbl@captionslist\@empty
2060   \else
2061     \bbl@read@ini{\bbl@KVP@captions}{data}% Here all letters cat = 11
2062     \bbl@after@ini
2063     \bbl@savestrings
2064   \fi
2065 \StartBabelCommands*{#1}{date}%
2066   \ifx\bbl@KVP@import\@nil
2067     \bbl@exp{%
2068       \\SetString\\today{\bbl@nocaption{today}{#1today}}}%
2069   \else
2070     \bbl@savetoday
2071     \bbl@savedate
2072   \fi
2073 \bbl@endcommands
2074 \bbl@exp{%
2075   \def\<#1hyphenmins>{%
2076     {\bbl@ifunset{\bbl@lfthm#1}{2}{\bbl@cs{lfthm#1}}}%
2077     {\bbl@ifunset{\bbl@rgthm#1}{3}{\bbl@cs{rgthm#1}}}}}%
2078 \bbl@provide@hyphens{#1}%
2079 \ifx\bbl@KVP@main\@nil\else
2080   \expandafter\main@language\expandafter{#1}%

```

```

2081 \fi}
2082 \def\bbl@provide@renew#1{%
2083 \ifx\bbl@KVP@captions\@nil\else
2084 \StartBabelCommands*{#1}{captions}%
2085 \bbl@read@ini{\bbl@KVP@captions}{data}% Here all letters cat = 11
2086 \bbl@after@ini
2087 \bbl@savestrings
2088 \EndBabelCommands
2089 \fi
2090 \ifx\bbl@KVP@import\@nil\else
2091 \StartBabelCommands*{#1}{date}%
2092 \bbl@savetoday
2093 \bbl@savedate
2094 \EndBabelCommands
2095 \fi
2096 % == hyphenrules ==
2097 \bbl@provide@hyphens{#1}}

```

The hyphenrules option is handled with an auxiliary macro.

```

2098 \def\bbl@provide@hyphens#1{%
2099 \let\bbl@tempa\relax
2100 \ifx\bbl@KVP@hyphenrules\@nil\else
2101 \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2102 \bbl@foreach\bbl@KVP@hyphenrules{%
2103 \ifx\bbl@tempa\relax % if not yet found
2104 \bbl@ifsamestring{##1}{+}%
2105 {\bbl@exp{\addlanguage\<l@##1>}}}%
2106 {}%
2107 \bbl@ifunset{l@##1}%
2108 {}%
2109 {\bbl@exp{\let\bbl@tempa\<l@##1>}}}%
2110 \fi}%
2111 \fi
2112 \ifx\bbl@tempa\relax % if no opt or no language in opt found
2113 \ifx\bbl@KVP@import\@nil\else % if importing
2114 \bbl@exp{% and hyphenrules is not empty
2115 \bbl@ifblank{\bbl@cs{hyphr@#1}}%
2116 {}%
2117 {\let\bbl@tempa\<l@bbl@c1{hyphr}>}}%
2118 \fi
2119 \fi
2120 \bbl@ifunset{\bbl@tempa}% ie, relax or undefined
2121 {\bbl@ifunset{l@#1}% no hyphenrules found - fallback
2122 {\bbl@exp{\adddialect\<l@#1>\language}}%
2123 {}}% so, l@<lang> is ok - nothing to do
2124 {\bbl@exp{\adddialect\<l@#1>\bbl@tempa}}}% found in opt list or ini
2125

```

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.

```

2126 \ifx\bbl@readstream\undefined
2127 \csname newread\endcsname\bbl@readstream
2128 \fi
2129 \def\bbl@inipreread#1=#2\@@{%
2130 \bbl@trim@def\bbl@tempa{#1}% Redundant below !!
2131 \bbl@trim\toks@{#2}%
2132 % Move trims here ??
2133 \bbl@ifunset{\bbl@KVP@\bbl@section/\bbl@tempa}%
2134 {\bbl@exp{%

```

```

2135      \\g@addto@macro\\bbl@inidata{%
2136      \\bbl@elt{\\bbl@section}{\\bbl@tempa}{\\the\\toks@}}}%
2137      \\expandafter\\bbl@inireader\\bbl@tempa=#2\\@@}%
2138      {}}%
2139 \\def\\bbl@read@ini#1#2{%
2140   \\global\\namedef{\\bbl@lini@\\language\\name}{#1}%
2141   \\openin\\bbl@readstream=babel-#1.ini
2142   \\ifeof\\bbl@readstream
2143     \\bbl@error
2144     {There is no ini file for the requested language\\%
2145     (#1). Perhaps you misspelled it or your installation\\%
2146     is not complete.}%
2147     {Fix the name or reinstall babel.}%
2148   \\else
2149     \\def\\bbl@inidata{\\bbl@elt{\\bbl@identificacion}{tag.ini}{#1}}}%
2150     \\let\\bbl@section\\@empty
2151     \\let\\bbl@savestrings\\@empty
2152     \\let\\bbl@savetoday\\@empty
2153     \\let\\bbl@savestate\\@empty
2154     \\let\\bbl@inireader\\bbl@iniskip
2155     \\bbl@info{Importing #2 for \\language\\name\\%
2156     from babel-#1.ini. Reported}%
2157     \\loop
2158     \\if T\\ifeof\\bbl@readstream F\\fi T\\relax % Trick, because inside \\loop
2159     \\endlinechar\\m@ne
2160     \\read\\bbl@readstream to \\bbl@line
2161     \\endlinechar`^^M
2162     \\ifx\\bbl@line\\@empty\\else
2163       \\expandafter\\bbl@iniline\\bbl@line\\bbl@iniline
2164     \\fi
2165     \\repeat
2166     \\bbl@foreach\\bbl@renewlist{%
2167       \\bbl@ifunset{\\bbl@renew@##1}{\\bbl@inisec[##1]\\@@}}%
2168     \\global\\let\\bbl@renewlist\\@empty
2169     % Ends last section. See \\bbl@inisec
2170     \\def\\bbl@elt##1##2{\\bbl@inireader##1=##2\\@@}%
2171     \\bbl@cs{renew@\\bbl@section}%
2172     \\global\\bbl@csarg\\let{renew@\\bbl@section}\\relax
2173     \\bbl@cs{secpost@\\bbl@section}%
2174     \\bbl@csarg{\\global\\expandafter\\let}{inidata@\\language\\name}\\bbl@inidata
2175     \\bbl@exp{\\bbl@add@list\\bbl@ini@loaded{\\language\\name}}%
2176     \\bbl@tglobal\\bbl@ini@loaded
2177     % \\bbl@exp{\\gdef\\<bbl@inidata@tag@#1>{\\<bbl@inidata@\\language\\name>}}%
2178     \\fi}
2179 \\def\\bbl@iniline#1\\bbl@iniline{%
2180   \\@ifnextchar[\\bbl@inisec{\\@ifnextchar;\\bbl@iniskip\\bbl@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.

```

2181 \\def\\bbl@iniskip#1\\@@{%      if starts with ;
2182 \\def\\bbl@inisec[#1]#2\\@@{%  if starts with opening bracket
2183   \\def\\bbl@elt##1##2{%
2184     \\expandafter\\toks@\\expandafter{%
2185       \\expandafter{\\bbl@section}{##1}{##2}}%
2186     \\bbl@exp{%
2187       \\g@addto@macro\\bbl@inidata{\\bbl@elt\\the\\toks@}}%

```

```

2188 \bbl@inireader##1=##2\@@}%
2189 \bbl@cs{renew@\bbl@section}%
2190 \global\bbl@csarg\let{renew@\bbl@section}\relax
2191 \bbl@cs{secpost@\bbl@section}%
2192 % The previous code belongs to the previous section.
2193 % Now start the current one.
2194 \def\bbl@section{#1}%
2195 \def\bbl@elt##1##2{%
2196   \@namedef{bbl@KVP@#1/##1}{}}%
2197 \bbl@cs{renew@#1}%
2198 \bbl@cs{secpre@#1}% pre-section 'hook'
2199 \bbl@ifunset{bbl@inikv@#1}%
2200   {\let\bbl@inireader\bbl@iniskip}%
2201   {\bbl@exp{\let\\bbl@inireader<bbl@inikv@#1>}}}%
2202 \let\bbl@renewlist@empty
2203 \def\bbl@renewinikv#1/#2\@@#3{%
2204   \bbl@ifunset{bbl@renew@#1}%
2205     {\bbl@add@list\bbl@renewlist{#1}}%
2206     {}}%
2207 \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>.

```

2208 \def\bbl@inikv#1=#2\@@{%      key=value
2209   \bbl@trim@def\bbl@tempa{#1}%
2210   \bbl@trim\toks@{#2}%
2211   \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.

```

2212 \def\bbl@exportkey#1#2#3{%
2213   \bbl@ifunset{bbl@kv@#2}%
2214     {\bbl@csarg\gdef{#1@\language}\{#3}}%
2215     {\expandafter\ifx\csname bbl@kv@#2\endcsname\empty
2216       \bbl@csarg\gdef{#1@\language}\{#3}}%
2217     \else
2218       \bbl@exp{\global\let<bbl@#1@\language>\<bbl@kv@#2>}}%
2219     \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@inise), while \bbl@after@ini must be called explicitly after \bbl@read@ini if necessary.

```

2220 \def\bbl@iniwarning#1{%
2221   \bbl@ifunset{bbl@kv@identification.warning#1}{}}%
2222   {\bbl@warning{%
2223     From babel-\bbl@cs{lini@\language}.ini:\%
2224     \bbl@cs{kv@identification.warning#1}\%
2225     Reported }}}
2226 \let\bbl@inikv@identification\bbl@inikv
2227 \def\bbl@secpost@identification{%
2228   \bbl@iniwarning}%
2229   \ifcase\bbl@engine
2230     \bbl@iniwarning{.pdf\latex}%
2231   \or
2232     \bbl@iniwarning{.lua\latex}%
2233   \or
2234     \bbl@iniwarning{.xel\latex}%
2235   \fi%
2236   \bbl@exportkey{elname}{identification.name.english}{}}%

```



```

2237 \bbl@exp{\bbbl@exportkey{lname}{identification.name.opentype}%
2238 {\csname bbl@elname@languagename\endcsname}}%
2239 \bbl@exportkey{lbcpr}{identification.tag.bcp47}{}%
2240 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2241 \bbl@exportkey{esname}{identification.script.name}{}%
2242 \bbl@exp{\bbbl@exportkey{sname}{identification.script.name.opentype}%
2243 {\csname bbl@esname@languagename\endcsname}}%
2244 \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
2245 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}}
2246 \let\bbl@inikv@typography\bbl@inikv
2247 \let\bbl@inikv@characters\bbl@inikv
2248 \let\bbl@inikv@numbers\bbl@inikv
2249 \def\bbl@inikv@counters#1=#2\@{%
2250 \def\bbl@tempc{#1}%
2251 \bbl@trim@def{\bbl@tempb*}{#2}%
2252 \in@{.1$}{#1$}%
2253 \ifin@
2254 \bbl@replace\bbl@tempc{.1}{}%
2255 \bbl@csarg\xdef{cntr@bbl@tempc @languagename}{%
2256 \noexpand\bbl@alphanumeric{\bbl@tempc}}%
2257 \fi
2258 \in@{.F.}{#1}%
2259 \ifin@else\in@{.S.}{#1}\fi
2260 \ifin@
2261 \bbl@csarg\xdef{cntr@#1@languagename}{\bbl@tempb*}%
2262 \else
2263 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
2264 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
2265 \bbl@csarg{\global\expandafter\let}{cntr@#1@languagename}\bbl@tempa
2266 \fi}
2267 \def\bbl@after@ini{%
2268 \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
2269 \bbl@exportkey{rgtm}{typography.righthyphenmin}{3}%
2270 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2271 \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2272 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2273 \bbl@exportkey{hyoth}{typography.hyphenate.other}{}%
2274 \bbl@exportkey{intsp}{typography.intraspaces}{}%
2275 \bbl@exportkey{jstfy}{typography.justify}{w}%
2276 \bbl@exportkey{chrng}{characters.ranges}{}%
2277 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2278 \bbl@exportkey{rqtex}{identification.require.babel}{}%
2279 \bbl@toglobal\bbl@savetoday
2280 \bbl@toglobal\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.

```

2281 \ifcase\bbl@engine
2282 \bbl@csarg\def{inikv@captions.licr}#1=#2\@{%
2283 \bbl@ini@captions@aux{#1}{#2}}
2284 \else
2285 \def\bbl@inikv@captions#1=#2\@{%
2286 \bbl@ini@captions@aux{#1}{#2}}
2287 \fi

```

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

```

2288 \def\bbl@ini@captions@aux#1#2{%
2289 \bbl@trim@def\bbl@tempa{#1}%

```

```

2290 \bbl@ifblank{#2}%
2291 {\bbl@exp{%
2292 \toks@{\bbl@nocaption{\bbl@tempa}{\language\language\bbl@tempa name}}}%
2293 {\bbl@trim\toks@{#2}}}%
2294 \bbl@exp{%
2295 \bbl@add\bbl@savestrings{%
2296 \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 copypaste pattern.

```

2297 \bbl@csarg\def{inikv@date.gregorian}#1=#2\@@{% for defaults
2298 \bbl@inidate#1...\relax{#2}{}}
2299 \bbl@csarg\def{inikv@date.islamic}#1=#2\@@{%
2300 \bbl@inidate#1...\relax{#2}{islamic}}
2301 \bbl@csarg\def{inikv@date.hebrew}#1=#2\@@{%
2302 \bbl@inidate#1...\relax{#2}{hebrew}}
2303 \bbl@csarg\def{inikv@date.persian}#1=#2\@@{%
2304 \bbl@inidate#1...\relax{#2}{persian}}
2305 \bbl@csarg\def{inikv@date.indian}#1=#2\@@{%
2306 \bbl@inidate#1...\relax{#2}{indian}}
2307 \ifcase\bbl@engine
2308 \bbl@csarg\def{inikv@date.gregorian.licr}#1=#2\@@{% override
2309 \bbl@inidate#1...\relax{#2}{}}
2310 \bbl@csarg\def{secpre@date.gregorian.licr}{% discard uni
2311 \ifcase\bbl@engine\let\bbl@savestate\empty\fi}
2312 \fi
2313 % eg: 1=months, 2=wide, 3=1, 4=dummy
2314 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
2315 \bbl@trim@def\bbl@tempa{#1.#2}%
2316 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savestate
2317 {\bbl@trim@def\bbl@tempa{#3}%
2318 \bbl@trim\toks@{#5}%
2319 \bbl@exp{%
2320 \bbl@add\bbl@savestate{%
2321 \SetString\month\romannumeral\bbl@tempa#6name>{\the\toks@}}}%
2322 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
2323 {\bbl@trim@def\bbl@toreplace{#5}%
2324 \bbl@TG@date
2325 \global\bbl@csarg\let{date@\language}\bbl@toreplace
2326 \bbl@exp{%
2327 \gdef\<\language date>{\protect\<\language date >}%
2328 \gdef\<\language date >####1####2####3{%
2329 \bbl@usedategroupttrue
2330 \<\bbl@ensure@\language>{%
2331 \<\bbl@date@\language>{####1}{####2}{####3}}}%
2332 \bbl@add\bbl@savetoday{%
2333 \SetString\today{%
2334 \<\language date>{\the\year}{\the\month}{\the\day}}}%
2335 }}

```

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.

```

2336 \let\bbl@calendar\empty
2337 \newcommand\BabelDateSpace{\nobreakspace}
2338 \newcommand\BabelDateDot{.\@}
2339 \newcommand\BabelDated[1]{\number#1}

```

```

2340 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
2341 \newcommand\BabelDateM[1]{\number#1}
2342 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
2343 \newcommand\BabelDateMMM[1]{%
2344 \csname month\romannumeral#1\bb1@calendar name\endcsname}%
2345 \newcommand\BabelDatey[1]{\number#1}%
2346 \newcommand\BabelDateyy[1]{%
2347 \ifnum#1<10 0\number#1 %
2348 \else\ifnum#1<100 \number#1 %
2349 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
2350 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
2351 \else
2352 \bb1@error
2353 {Currently two-digit years are restricted to the\
2354 range 0-9999.}%
2355 {There is little you can do. Sorry.}%
2356 \fi\fi\fi\fi}
2357 \newcommand\BabelDateyyyy[1]{\number#1} % FIXME - add leading 0
2358 \def\bb1@replace@finish@iii#1{%
2359 \bb1@exp{\def\#1####1####2####3{\the\toks@}}
2360 \def\bb1@TG@date{%
2361 \bb1@replace\bb1@toreplace{[ ]}{\BabelDateSpace{}}%
2362 \bb1@replace\bb1@toreplace{[.]}{\BabelDateDot{}}%
2363 \bb1@replace\bb1@toreplace{[d]}{\BabelDated{####3}}%
2364 \bb1@replace\bb1@toreplace{[dd]}{\BabelDatedd{####3}}%
2365 \bb1@replace\bb1@toreplace{[M]}{\BabelDateM{####2}}%
2366 \bb1@replace\bb1@toreplace{[MM]}{\BabelDateMM{####2}}%
2367 \bb1@replace\bb1@toreplace{[MMM]}{\BabelDateMMM{####2}}%
2368 \bb1@replace\bb1@toreplace{[y]}{\BabelDatey{####1}}%
2369 \bb1@replace\bb1@toreplace{[yy]}{\BabelDateyy{####1}}%
2370 \bb1@replace\bb1@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
2371 % Note after \bb1@replace \toks@ contains the resulting string.
2372 % TODO - Using this implicit behavior doesn't seem a good idea.
2373 \bb1@replace@finish@iii\bb1@toreplace}

```

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

```

2374 \def\bb1@provide@lsys#1{%
2375 \bb1@ifunset{bb1@lname@#1}%
2376 {\bb1@ini@basic{#1}}%
2377 {}}
2378 \bb1@csarg\let{lsys@#1}\@empty
2379 \bb1@ifunset{bb1@sname@#1}{\bb1@csarg\gdef{sname@#1}{Default}}}%
2380 \bb1@ifunset{bb1@sotf@#1}{\bb1@csarg\gdef{sotf@#1}{DFLT}}}%
2381 \bb1@csarg\bb1@add@list{lsys@#1}{Script=\bb1@cs{sname@#1}}%
2382 \bb1@ifunset{bb1@lname@#1}{%
2383 {\bb1@csarg\bb1@add@list{lsys@#1}{Language=\bb1@cs{lname@#1}}}%
2384 \ifcase\bb1@engine\or\or
2385 \bb1@ifunset{bb1@prehc@#1}{%
2386 {\bb1@exp{\bb1@ifblank{\bb1@cs{prehc@#1}}}%
2387 {}}%
2388 {\bb1@csarg\bb1@add@list{lsys@#1}{HyphenChar="200B}}}%
2389 \fi
2390 \bb1@csarg\bb1@to@global{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.

```

2391 \def\bbl@ini@basic#1{%
2392   \def\BabelBeforeIni###1##2{%
2393     \begingroup
2394       \bbl@add\bbl@secpost@identification{\closein\bbl@readstream}%
2395       \catcode\ [=12 \catcode\ ]=12 \catcode\ =12 \catcode\ ;=12%
2396       \bbl@read@ini{##1}{font and identification data}%
2397       \endinput           % babel- .tex may contain onlypreamble's
2398       \endgroup}%         boxed, to avoid extra spaces:
2399   {\setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}}{}}

```

Alphabetic counters must be converted from a space separated list to an \ifcase structure.

```

2400 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={%
2401   \ifx\#1%                % \ before, in case #1 is multiletter
2402     \bbl@exp{%
2403       \def\ \bbl@tempa####1{%
2404         \ifcase>####1\space\the\toks@<else>\ \@ctrerr<fi>}}%
2405     \else
2406       \toks@\expandafter{\the\toks@ \or #1}%
2407       \expandafter\bbl@buildifcase
2408     \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).

```

2409 \newcommand\localnumeral[2]{\bbl@cs{cntr@#1@\language}\{#2}}
2410 \def\bbl@localecntr#1#2{\localnumeral{#2}{#1}}
2411 \newcommand\localecounter[2]{%
2412   \expandafter\bbl@localecntr\csname c@#2\endcsname{#1}}
2413 \def\bbl@alphanumeric#1#2{%
2414   \expandafter\bbl@alphanumeric@i\number#2 76543210\@@{#1}}
2415 \def\bbl@alphanumeric@i#1#2#3#4#5#6#7#8\@@#9{%
2416   \ifcase\car#8\@nil\or    % Currenty <10000, but prepared for bigger
2417     \bbl@alphanumeric@ii{#9}000000#1\or
2418     \bbl@alphanumeric@ii{#9}00000#1#2\or
2419     \bbl@alphanumeric@ii{#9}0000#1#2#3\or
2420     \bbl@alphanumeric@ii{#9}000#1#2#3#4\else
2421     \bbl@alphanum@invalid{>9999}%
2422   \fi}
2423 \def\bbl@alphanumeric@ii#1#2#3#4#5#6#7#8{%
2424   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\language}%
2425     {\bbl@cs{cntr@#1.4@\language}\{#5}}
2426     {\bbl@cs{cntr@#1.3@\language}\{#6}}
2427     {\bbl@cs{cntr@#1.2@\language}\{#7}}
2428     {\bbl@cs{cntr@#1.1@\language}\{#8}}
2429     \ifnum#6#7#8>\z@ % An ad hoc rule for Greek. Ugly. To be fixed.
2430       \bbl@ifunset{bbl@cntr@#1.S.321@\language}\{#9}}
2431       {\bbl@cs{cntr@#1.S.321@\language}\{#9}}
2432   \fi}%
2433   {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}}
2434 \def\bbl@alphanum@invalid#1{%
2435   \bbl@error{Alphabetic numeral too large (#1)}%
2436   {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.

```

2437 \newcommand\localeinfo[1]{%
2438   \bbl@ifunset{bbl@csname bbl@info@#1\endcsname @\language}%
2439   {\bbl@error{I've found no info for the current locale.\%
2440     The corresponding ini file has not been loaded\%
2441     Perhaps it doesn't exist}%
2442     {See the manual for details.}}%
2443   {\bbl@cs{csname bbl@info@#1\endcsname @\language}}
2444 % \@namedef{bbl@info@name.locale}{lname}
2445 \@namedef{bbl@info@tag.ini}{lini}
2446 \@namedef{bbl@info@name.english}{elname}
2447 \@namedef{bbl@info@name.opentype}{lname}
2448 \@namedef{bbl@info@tag.bcp47}{lbcpr}
2449 \@namedef{bbl@info@tag.opentype}{lotf}
2450 \@namedef{bbl@info@script.name}{esname}
2451 \@namedef{bbl@info@script.name.opentype}{sname}
2452 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
2453 \@namedef{bbl@info@script.tag.opentype}{sotf}
2454 \let\bbl@ensureinfo\@gobble
2455 \newcommand\BabelEnsureInfo{%
2456   \def\bbl@ensureinfo##1{%
2457     \ifx\InputIfFileExists\undefined\else % not in plain
2458       \bbl@ifunset{bbl@lname@##1}{\bbl@ini@basic{##1}}{}%
2459       \fi}}

```

More general, but non-expandable, is `\getlocaledata`. 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`.

```

2460 \newcommand\getlocaledata[3]{%
2461   \let#1\relax
2462   \def\bbl@elt##1##2##3{%
2463     \bbl@ifsamestring{##1/##2}{##3}%
2464     {\providecommand#1{##3}%
2465     \def\bbl@elt####1####2####3{}}%
2466     {}}%
2467   \bbl@cs{inidata@#2}%
2468   \ifx#1\relax
2469     \bbl@error
2470     {Unknown key for locale '#2':\%
2471     #3\%
2472     \string#1 will be set to \relax}%
2473     {Perhaps you misspelled it.}%
2474   \fi}
2475 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

## 10 Adjusting the Babel behavior

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

```

2476 \newcommand\babeladjust[1]{% TODO. Error handling.
2477   \bbl@forkv{#1}{\bbl@cs{ADJ@##1@##2}}
2478 %
2479 \def\bbl@adjust@lua#1#2{%
2480   \ifvmode
2481     \ifnum\currentgrouplevel=\z@
2482       \directlua{ Babel.#2 }%
2483       \expandafter\expandafter\expandafter\@gobble
2484     \fi
2485   \fi

```

```

2486 {\bbl@error % The error is gobbled if everything went ok.
2487 {Currently, #1 related features can be adjusted only\\%
2488 in the main vertical list.}%
2489 {Maybe things change in the future, but this is what it is.}}}
2490 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
2491 \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
2492 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
2493 \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
2494 \@namedef{bbl@ADJ@bidi.text@on}{%
2495 \bbl@adjust@lua{bidi}{bidi_enabled=true}}
2496 \@namedef{bbl@ADJ@bidi.text@off}{%
2497 \bbl@adjust@lua{bidi}{bidi_enabled=false}}
2498 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
2499 \bbl@adjust@lua{bidi}{digits_mapped=true}}
2500 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
2501 \bbl@adjust@lua{bidi}{digits_mapped=false}}
2502 %
2503 \@namedef{bbl@ADJ@linebreak.sea@on}{%
2504 \bbl@adjust@lua{linebreak}{sea_enabled=true}}
2505 \@namedef{bbl@ADJ@linebreak.sea@off}{%
2506 \bbl@adjust@lua{linebreak}{sea_enabled=false}}
2507 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
2508 \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
2509 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
2510 \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
2511 %
2512 \def\bbl@adjust@layout#1{%
2513 \ifvmode
2514 #1%
2515 \expandafter\@gobble
2516 \fi
2517 {\bbl@error % The error is gobbled if everything went ok.
2518 {Currently, layout related features can be adjusted only\\%
2519 in vertical mode.}%
2520 {Maybe things change in the future, but this is what it is.}}}
2521 \@namedef{bbl@ADJ@layout.tabular@on}{%
2522 \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
2523 \@namedef{bbl@ADJ@layout.tabular@off}{%
2524 \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
2525 \@namedef{bbl@ADJ@layout.lists@on}{%
2526 \bbl@adjust@layout{\let\list\bbl@NL@list}}
2527 \@namedef{bbl@ADJ@layout.lists@off}{%
2528 \bbl@adjust@layout{\let\list\bbl@OL@list}}
2529 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
2530 \bbl@activateposthyphen}

```

## 11 The kernel of Babel (babel.def for L<sup>A</sup>T<sub>E</sub>X only)

### 11.1 The redefinition of the style commands

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

```

2531 {\def\format{lplain}

```

```

2532 \ifx\fmtname\format
2533 \else
2534   \def\format{LaTeX2e}
2535   \ifx\fmtname\format
2536   \else
2537     \aftergroup\endinput
2538   \fi
2539 \fi}

```

## 11.2 Cross referencing macros

The  $\LaTeX$  book states:

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

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

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

The only way to accomplish this in most cases is to use the trick described in the  $\TeX$ book [4] (Appendix D, page 382). The primitive `\meaning` applied to a token expands to the current meaning of this token. For example, ‘`\meaning\A`’ with `\A` defined as ‘`\def\A#1{\B}`’ expands to the characters ‘`macro:#1->\B`’ with all category codes set to ‘other’ or ‘space’.

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

```

2540 %\bbl@redefine\newlabel#1#2{%
2541 %  \@safe@activetrue\org@newlabel{#1}{#2}\@safe@activfalse}

```

`\@newl@bel` We need to change the definition of the  $\LaTeX$ -internal macro `\@newl@bel`. This is needed because we need to make sure that shorthand characters expand to their non-active version.

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

```

2542 <<(*More package options)>> ≡
2543 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
2544 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
2545 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
2546 <</More package options>>

```

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

```

2547 \bbl@trace{Cross referencing macros}
2548 \ifx\bbl@opt@safe\@empty\else
2549   \def\@newl@bel#1#2#3{%
2550     {\@safe@activetrue
2551       \bbl@ifunset{#1@#2}%
2552         \relax
2553       {\gdef\@multiplelabels{%
2554         \@latex@warning@no@line{There were multiply-defined labels}}%
2555       \@latex@warning@no@line{Label `#2' multiply defined}}%
2556     \global\@namedef{#1@#2}{#3}}

```

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

```
2557 \CheckCommand*\@testdef[3]{%
2558   \def\reserved@a{#3}%
2559   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
2560   \else
2561     \@tempswatrue
2562   \fi}
```

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

```
2563 \def\@testdef#1#2#3{%
2564   \@safe@activetrue
```

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

```
2565   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
```

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

```
2566   \def\bbl@tempb{#3}%
2567   \@safe@activesfalse
```

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

```
2568   \ifx\bbl@tempa\relax
2569   \else
2570     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
2571   \fi
```

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

```
2572   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
```

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

```
2573   \ifx\bbl@tempa\bbl@tempb
2574   \else
2575     \@tempswatrue
2576   \fi}
2577 \fi
```

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

```
2578 \bbl@xin@{R}\bbl@opt@safe
2579 \ifin@
2580   \bbl@redefineroobust\ref#1{%
2581     \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
2582   \bbl@redefineroobust\pageref#1{%
2583     \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
2584 \else
2585   \let\org@ref\ref
2586   \let\org@pageref\pageref
2587 \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.

```
2588 \bbl@xin@{B}\bbl@opt@safe
2589 \ifin@
2590 \bbl@redefine\@citex[#1]#2{%
2591   \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
2592   \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.

```
2593 \AtBeginDocument{%
2594   \@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.)

```
2595   \def\@citex[#1][#2]#3{%
2596     \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
2597     \org@@citex[#1][#2]{\@tempa}}%
2598   }{}}
```

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

```
2599 \AtBeginDocument{%
2600   \@ifpackageloaded{cite}{%
2601     \def\@citex[#1]#2{%
2602       \@safe@activetrue\org@@citex[#1]{#2}\@safe@activesfalse}%
2603     }{}}
```

`\nocite` The macro `\nocite` which is used to instruct `BiBTX` to extract uncited references from the database.

```
2604 \bbl@redefine\nocite#1{%
2605   \@safe@activetrue\org@nocite{#1}\@safe@activesfalse}
```

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

```
2606 \bbl@redefine\bibcite{%
2607   \bbl@cite@choice
2608   \bibcite}
```

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

```
2609 \def\bbl@bibcite#1#2{%
2610   \org@bibcite{#1}{\@safe@activesfalse#2}}
```

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

```
2611 \def\bbl@cite@choice{%
2612   \global\let\bibcite\bbl@bibcite
```

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

```
2613   \ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
2614   \ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
```

Make sure this only happens once.

```
2615   \global\let\bbl@cite@choice\relax}
```

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

```
2616   \AtBeginDocument{\bbl@cite@choice}
```

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

```
2617 \bbl@redefine\@bibitem#1{%
2618   \@safe@activetrue\org@@bibitem{#1}\@safe@activetruefalse}
2619 \else
2620   \let\org@nocite\nocite
2621   \let\org@@citex\@citex
2622   \let\org@bibcite\bibcite
2623   \let\org@@bibitem\@bibitem
2624 \fi
```

### 11.3 Marks

`\markright` Because the output routine is asynchronous, we must pass the current language attribute to the head lines, together with the text that is put into them. To achieve this we need to adapt the definition of `\markright` and `\markboth` somewhat.

We check whether the argument is empty; if it is, we just make sure the scratch token register is empty. Next, we store the argument to `\markright` in the scratch token register. This way these commands will not be expanded later, and we make sure that the text is typeset using the correct language settings. While doing so, we make sure that active characters that may end up in the mark are not disabled by the output routine kicking in while `\@safe@activetrue` is in effect.

```
2625 \bbl@trace{Marks}
2626 \IfBabelLayout{sectioning}
2627   {\ifx\bbl@opt@headfoot\@nnil
2628     \g@addto@macro\@resetactivechars{%
2629       \set@typeset@protect
2630       \expandafter\select@language@x\expandafter{\bbl@main@language}%
2631       \let\protect\noexpand
2632       \edef\thepage{%
2633         \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
2634     \fi}
2635   {\ifbbl@single\else
2636     \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
2637     \markright#1{%
2638       \bbl@ifblank{#1}%
2639       {\org@markright}}}%
2640     {\toks@{#1}}%
2641     \bbl@exp{%
```

```

2642          \org@markright{\protect\foreignlanguage{\language}%
2643          {\protect\bb@l@restore@actives\the\toks@}}}%

```

`\markboth` The definition of `\markboth` is equivalent to that of `\markright`, except that we need two token registers. The documentclasses `report` and `book` define and set the headings for the page. While doing so they also store a copy of `\markboth` in `\@mkboth`. Therefore we need to check whether `\@mkboth` has already been set. If so we need to do that again with the new definition of `\markboth`. (As of Oct 2019,  $\LaTeX$  stores the definition in an intermediate macros, so it's not necessary anymore, but it's preserved for older versions.)

```

2644      \ifx\@mkboth\markboth
2645      \def\bb@l@tempc{\let\@mkboth\markboth}
2646      \else
2647      \def\bb@l@tempc{}
2648      \fi
2649      \bb@l@ifunset{markboth }{\bb@l@redefine\bb@l@redefineroobust
2650      \markboth#1#2{%
2651      \protected@edef\bb@l@tempb##1{%
2652      \protect\foreignlanguage
2653      {\language}%{\protect\bb@l@restore@actives##1}}%
2654      \bb@l@ifblank{#1}%
2655      {\toks@{}}%
2656      {\toks@\expandafter{\bb@l@tempb{#1}}}%
2657      \bb@l@ifblank{#2}%
2658      {\@temptokena{}}%
2659      {\@temptokena\expandafter{\bb@l@tempb{#2}}}%
2660      \bb@l@exp{\org@markboth{\the\toks@}{\the\@temptokena}}}
2661      \bb@l@tempc
2662      \fi} % end ifbb@l@single, end \IfBabelLayout

```

## 11.4 Preventing clashes with other packages

### 11.4.1 `ifthen`

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

```

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

```

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

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

```

2663 \bb@l@trace{Preventing clashes with other packages}
2664 \bb@l@xin@{R}\bb@l@opt@safe
2665 \ifin@
2666 \AtBeginDocument{%
2667 \ifpackageloaded{ifthen}{%

```

Then we can redefine `\ifthenelse`:

```

2668 \bb@l@redefine@long\ifthenelse#1#2#3{%

```

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

```

2669      \let\bbl@temp@pref\pageref
2670      \let\pageref\org@pageref
2671      \let\bbl@temp@ref\ref
2672      \let\ref\org@ref

```

Then we can set the `\@safe@activestrue` switch and call the original `\ifthenelse`. In order to be able to use shorthands in the second and third arguments of `\ifthenelse` the resetting of the switch *and* the definition of `\pageref` happens inside those arguments. When the package wasn't loaded we do nothing.

```

2673      \@safe@activestrue
2674      \org@ifthenelse{#1}%
2675      {\let\pageref\bbl@temp@pref
2676       \let\ref\bbl@temp@ref
2677       \@safe@activestrue
2678       #2}%
2679      {\let\pageref\bbl@temp@pref
2680       \let\ref\bbl@temp@ref
2681       \@safe@activestrue
2682       #3}%
2683      }%
2684      }{}%
2685      }

```

#### 11.4.2 varioref

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

```

2686 \AtBeginDocument{%
2687   \ifpackageloaded{varioref}{%
2688     \bbl@redefine\@@vpageref#1[#2]#3{%
2689       \@safe@activestrue
2690       \org@@@vpageref{#1}[#2]#3}%
2691     \@safe@activestrue}%
2692   \bbl@redefine\vrefpagemum#1#2{%
2693     \@safe@activestrue
2694     \org@vrefpagemum{#1}#2}%
2695   \@safe@activestrue}%

```

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.

```

2696   \expandafter\def\csname Ref \endcsname#1{%
2697     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
2698   }{}%
2699   }
2700 \fi

```

#### 11.4.3 hhline

`\hhline` Delaying the activation of the shorthand characters has introduced a problem with the `hhline` package. The reason is that it uses the `‘` character which is made active by the french support in `babel`. Therefore we need to *reload* the package when the `‘` is an active character.

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

```
2701 \AtEndOfPackage{%
2702   \AtBeginDocument{%
2703     \ifpackageloaded{hhline}%
```

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

```
2704     {\expandafter\ifx\csname normal@char\string\endcsname\relax
2705       \else
```

In that case we simply reload the package. Note that this happens *after* the category code of the `@`-sign has been changed to other, so we need to temporarily change it to letter again.

```
2706         \makeatletter
2707         \def\@currname{hhline}\input{hhline.sty}\makeatother
2708         \fi}%
2709       {}}}
```

#### 11.4.4 hyperref

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

```
2710 \AtBeginDocument{%
2711   \ifx\pdfstringdefDisableCommands\undefined\else
2712     \pdfstringdefDisableCommands{\languageshortands{system}}%
2713   \fi}
```

#### 11.4.5 fancyhdr

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

```
2714 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
2715   \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.

```
2716 \def\substitutefontfamily#1#2#3{%
2717   \lowercase{\immediate\openout15=#1#2.fd\relax}%
2718   \immediate\write15{%
2719     \string\ProvidesFile{#1#2.fd}%
2720     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
2721     \space generated font description file]^J
2722     \string\DeclareFontFamily{#1}{#2}{^J
2723     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^J
2724     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^J
2725     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^J
2726     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^J
2727     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^J
2728     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^J
2729     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^J
2730     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^J
2731     }%
2732   \closeout15
2733 }
```

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

```
2734 \@onlypreamble\substitutefontfamily
```

## 11.5 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of  $\TeX$  and  $\LaTeX$  always come out in the right encoding. There is a list of non-ASCII encodings. Unfortunately, fontenc deletes its package options, so we must guess which encodings has been loaded by traversing `\@filelist` to search for `\enc.def`. If a non-ASCII has been loaded, we define versions of  $\TeX$  and  $\LaTeX$  for them using `\ensureascii`. The default ASCII encoding is set, too (in reverse order): the “main” encoding (when the document begins), the last loaded, or OT1.

`\ensureascii`

```
2735 \bbl@trace{Encoding and fonts}
2736 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,PU,PD1}
2737 \newcommand\BabelNonText{TS1,T3,TS3}
2738 \let\org@TeX\TeX
2739 \let\org@LaTeX\LaTeX
2740 \let\ensureascii\@firstofone
2741 \AtBeginDocument{%
2742   \in@false
2743   \bbl@foreach\BabelNonASCII{% is there a text non-ascii enc?
2744     \ifin@
2745       \lowercase{\bbl@xin@{,#1\enc.def,}\@filelist,}}%
2746     \fi}%
2747   \ifin@ % if a text non-ascii has been loaded
2748     \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
2749     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
2750     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
2751     \def\bbl@tempb#1\@{\uppercase{\bbl@tempc#1}ENC.DEF\empty\@}%
2752     \def\bbl@tempc#1ENC.DEF#2\@{\%
2753       \ifx\empty#2\else
2754         \bbl@ifunset{T#1}%
2755         {}%
2756         {\bbl@xin@{,#1,}\@,\BabelNonASCII,\BabelNonText,}%
2757         \ifin@
2758           \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
2759           \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
2760         \else
2761           \def\ensureascii##1{{\fontencoding{#1}\selectfont##1}}%
2762           \fi}%
2763       \fi}%
2764   \bbl@foreach\@filelist{\bbl@tempb#1\@}% TODO - \@ de mas??
2765   \bbl@xin@{\cf@encoding,}\@,\BabelNonASCII,\BabelNonText,%
2766   \ifin@
2767     \edef\ensureascii#1{{%
2768       \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
2769     \fi
2770   \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.

```
2771 \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.

```

2772 \AtBeginDocument{%
2773   \ifpackageloaded{fontspec}%
2774     {\xdef\latinencoding{%
2775       \ifx\UTFencname\@undefined
2776         EU\ifcase\bbl@engine\or2\or1\fi
2777       \else
2778         \UTFencname
2779       \fi}}%
2780   {\gdef\latinencoding{OT1}%
2781     \ifx\cf@encoding\bbl@t@one
2782       \xdef\latinencoding{\bbl@t@one}%
2783     \else
2784       \ifx\@fontenc@load@list\@undefined
2785         \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}{}%
2786       \else
2787         \def\@elt#1{, #1,}%
2788         \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
2789         \let\@elt\relax
2790         \bbl@xin@{, T1, }\bbl@tempa
2791         \ifin@
2792           \xdef\latinencoding{\bbl@t@one}%
2793         \fi
2794       \fi
2795     \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.

```

2796 \DeclareRobustCommand{\latintext}{%
2797   \fontencoding{\latinencoding}\selectfont
2798   \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.

```

2799 \ifx\@undefined\DeclareTextFontCommand
2800   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
2801 \else
2802   \DeclareTextFontCommand{\textlatin}{\latintext}
2803 \fi

```

## 11.6 Basic bidi support

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

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

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

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

```

2804 \bbl@trace{Basic (internal) bidi support}
2805 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
2806 \def\bbl@rscripts{%
2807   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
2808   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaean,%
2809   Manichaean,Meroitic Cursive,Meroitic,Old North Arabian,%
2810   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
2811   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
2812   Old South Arabian,}%
2813 \def\bbl@provide@dirs#1{%
2814   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
2815   \ifin@
2816     \global\bbl@csarg\chardef{wdir@#1}\@ne
2817     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
2818     \ifin@
2819       \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
2820     \fi
2821   \else
2822     \global\bbl@csarg\chardef{wdir@#1}\z@
2823   \fi
2824   \ifodd\bbl@engine
2825     \bbl@csarg\ifcase{wdir@#1}%
2826       \directlua{ Babel.locale_props[\the\localeid].texdir = 'l' }%
2827     \or
2828       \directlua{ Babel.locale_props[\the\localeid].texdir = 'r' }%
2829     \or
2830       \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
2831     \fi
2832   \fi}
2833 \def\bbl@switchdir{%
2834   \bbl@ifunset{bbl@sys@\languagename}{\bbl@provide@sys{\languagename}}{}%
2835   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
2836   \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}
2837 \def\bbl@setdirs#1{% TODO - math
2838   \ifcase\bbl@select@type % TODO - strictly, not the right test
2839     \bbl@bodydir{#1}%
2840     \bbl@pardir{#1}%
2841   \fi
2842   \bbl@texdir{#1}}
2843 \ifodd\bbl@engine % luatex=1
2844   \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2845   \DisableBabelHook{babel-bidi}
2846   \chardef\bbl@thetexdir\z@
2847   \chardef\bbl@thepardir\z@
2848   \def\bbl@getluadir#1{%
2849     \directlua{
2850       if tex.#1dir == 'TLT' then

```



```

2851         tex.sprint('0')
2852     elseif tex.#1dir == 'TRT' then
2853         tex.sprint('1')
2854     end}}
2855 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
2856     \ifcase#3\relax
2857         \ifcase\bbl@getluadir{#1}\relax\else
2858             #2 TLT\relax
2859         \fi
2860     \else
2861         \ifcase\bbl@getluadir{#1}\relax
2862             #2 TRT\relax
2863         \fi
2864     \fi}
2865 \def\bbl@textdir#1{%
2866     \bbl@setluadir{text}\textdir{#1}%
2867     \chardef\bbl@thetextdir#1\relax
2868     \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
2869 \def\bbl@pardir#1{%
2870     \bbl@setluadir{par}\pardir{#1}%
2871     \chardef\bbl@thepardir#1\relax}
2872 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
2873 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
2874 \def\bbl@dirparastext{\pardir\the\textdir\relax}% %%%
2875 % Sadly, we have to deal with boxes in math with basic.
2876 % Activated every math with the package option bidi=:
2877 \def\bbl@mathboxdir{%
2878     \ifcase\bbl@thetextdir\relax
2879         \everyhbox{\textdir TLT\relax}%
2880     \else
2881         \everyhbox{\textdir TRT\relax}%
2882     \fi}
2883 \else % pdftex=0, xetex=2
2884     \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2885     \DisableBabelHook{babel-bidi}
2886     \newcount\bbl@dirlevel
2887     \chardef\bbl@thetextdir\z@
2888     \chardef\bbl@thepardir\z@
2889     \def\bbl@textdir#1{%
2890         \ifcase#1\relax
2891             \chardef\bbl@thetextdir\z@
2892             \bbl@textdir@i\beginL\endL
2893         \else
2894             \chardef\bbl@thetextdir@ne
2895             \bbl@textdir@i\beginR\endR
2896         \fi}
2897     \def\bbl@textdir@i#1#2{%
2898         \ifhmode
2899             \ifnum\currentgrouplevel>\z@
2900                 \ifnum\currentgrouplevel=\bbl@dirlevel
2901                     \bbl@error{Multiple bidi settings inside a group}%
2902                     {I'll insert a new group, but expect wrong results.}%
2903                     \bgroup\aftergroup#2\aftergroup\egroup
2904                 \else
2905                     \ifcase\currentgrouptype\or % 0 bottom
2906                         \aftergroup#2% 1 simple {}
2907                     \or
2908                         \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
2909                     \or

```

```

2910      \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
2911      \or\or\or % vbox vtop align
2912      \or
2913      \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
2914      \or\or\or\or\or\or % output math disc insert vcent mathchoice
2915      \or
2916      \aftergroup#2% 14 \begingroup
2917      \else
2918      \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
2919      \fi
2920      \fi
2921      \bbl@dirlevel\currentgrouplevel
2922      \fi
2923      #1%
2924      \fi}
2925      \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
2926      \let\bbl@bodydir\@gobble
2927      \let\bbl@pagedir\@gobble
2928      \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).

```

2929      \def\bbl@xebidipar{%
2930      \let\bbl@xebidipar\relax
2931      \TeXeTstate\@ne
2932      \def\bbl@xeeverypar{%
2933      \ifcase\bbl@thepardir
2934      \ifcase\bbl@thetextdir\else\beginR\fi
2935      \else
2936      {\setbox\z@\lastbox\beginR\box\z@}%
2937      \fi}%
2938      \let\bbl@severypar\everypar
2939      \newtoks\everypar
2940      \everypar=\bbl@severypar
2941      \bbl@severypar{\bbl@xeeverypar\the\everypar}}
2942      \def\bbl@tempb{%
2943      \let\bbl@textdir\i\@gobbletwo
2944      \let\bbl@xebidipar\@empty
2945      \AddBabelHook{bidi}{foreign}{%
2946      \def\bbl@tempa{\def\BabelText#####1}%
2947      \ifcase\bbl@thetextdir
2948      \expandafter\bbl@tempa\expandafter{\BabelText{\LR{#####1}}}%
2949      \else
2950      \expandafter\bbl@tempa\expandafter{\BabelText{\RL{#####1}}}%
2951      \fi}
2952      \def\bbl@pardir##1{\ifcase##1\relax\setLR\else\setRL\fi}}
2953      \@ifpackagewith{babel}{bidi=bidi}{\bbl@tempb}{}%
2954      \@ifpackagewith{babel}{bidi=bidi-l}{\bbl@tempb}{}%
2955      \@ifpackagewith{babel}{bidi=bidi-r}{\bbl@tempb}{}%
2956      \fi

```

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

```

2957      \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
2958      \AtBeginDocument{%
2959      \ifx\pdfstringdefDisableCommands\@undefined\else
2960      \ifx\pdfstringdefDisableCommands\relax\else
2961      \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
2962      \fi

```

2963 \fi}

## 11.7 Local Language Configuration

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

```
2964 \bbl@trace{Local Language Configuration}
2965 \ifx\loadlocalcfg\@undefined
2966   \@ifpackagewith{babel}{noconfigs}%
2967   {\let\loadlocalcfg\@gobble}%
2968   {\def\loadlocalcfg#1{%
2969     \InputIfFileExists{#1.cfg}%
2970     {\typeout{*****^J%
2971               * Local config file #1.cfg used^^J%
2972               *}}%
2973     \@empty}}
2974 \fi
```

Just to be compatible with  $\text{\TeX}$  2.09 we add a few more lines of code:

```
2975 \ifx\@unexpandable@protect\@undefined
2976   \def\@unexpandable@protect{\noexpand\protect\noexpand}
2977   \long\def\protected@write#1#2#3{%
2978     \begingroup
2979       \let\thepage\relax
2980       #2%
2981       \let\protect\@unexpandable@protect
2982       \edef\reserved@a{\write#1{#3}}%
2983       \reserved@a
2984     \endgroup
2985     \if@nobreak\ifvmode\nobreak\fi\fi}
2986 \fi
2987 \</core>
2988 \<*kernel>
```

## 12 Multiple languages (switch.def)

Plain  $\text{\TeX}$  version 3.0 provides the primitive `\language` that is used to store the current language. When used with a pre-3.0 version this function has to be implemented by allocating a counter.

```
2989 \<<Make sure ProvidesFile is defined>>
2990 \ProvidesFile{switch.def}[\<<date>> \<<version>> Babel switching mechanism]
2991 \<<Load macros for plain if not LaTeX>>
2992 \<<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.

```
2993 \def\bbl@version{\<<version>>}
2994 \def\bbl@date{\<<date>>}
2995 \def\adddialect#1#2{%
2996   \global\chardef#1#2\relax
2997   \bbl@usehooks{adddialect}{\<#1>\<#2>}%
```

```

2998 \begingroup
2999 \count@#1\relax
3000 \def\bbl@elt##1##2##3##4{%
3001   \ifnum\count@=##2\relax
3002     \bbl@info{\string#1 = using hyphenrules for ##1\\%
3003       (\string\language\the\count@)}%
3004     \def\bbl@elt####1####2####3####4{%
3005       \fi}%
3006   \bbl@cs{languages}%
3007 \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.

```

3008 \def\bbl@fixname#1{%
3009   \begingroup
3010   \def\bbl@tempe{l@}%
3011   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
3012   \bbl@tempd
3013     {\lowercase\expandafter{\bbl@tempd}%
3014      {\uppercase\expandafter{\bbl@tempd}%
3015       \@empty
3016        {\edef\bbl@tempd{\def\noexpand#1{#1}}%
3017         \uppercase\expandafter{\bbl@tempd}}}%
3018       {\edef\bbl@tempd{\def\noexpand#1{#1}}%
3019        \lowercase\expandafter{\bbl@tempd}}}%
3020   \@empty
3021   \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
3022   \bbl@tempd
3023   \bbl@usehooks{language}{}}
3024 \def\bbl@iflanguage#1{%
3025   \@ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}

```

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

```

3026 \def\iflanguage#1{%
3027   \bbl@iflanguage{#1}%
3028   \ifnum\csname l@#1\endcsname=\language
3029     \expandafter\@firstoftwo
3030   \else
3031     \expandafter\@secondoftwo
3032   \fi}

```

## 12.1 Selecting the language

`\selectlanguage` The macro `\selectlanguage` checks whether the language is already defined before it performs its actual task, which is to update `\language` and activate language-specific definitions. To allow the call of `\selectlanguage` either with a control sequence name or with a simple string as argument, we have to use a trick to delete the optional escape character.

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

```
3033 \let\bbl@select@type\z@
3034 \edef\selectlanguage{%
3035   \noexpand\protect
3036   \expandafter\noexpand\csname selectlanguage \endcsname}
```

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

```
3037 \ifx\@undefined\protect\let\protect\relax\fi
```

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

```
3038 \ifx\documentclass\@undefined
3039   \def\xstring{\string\string\string}
3040 \else
3041   \let\xstring\string
3042 \fi
```

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

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

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

```
3043 \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  
`\bbl@pop@language` be simple:

```
3044 \def\bbl@push@language{%
3045   \xdef\bbl@language@stack{\languagename+\bbl@language@stack}}
```

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

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

```
3046 \def\bbl@pop@lang#1+#2-#3{%
3047   \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.

```
3048 \let\bbl@ifrestoring\@secondoftwo
3049 \def\bbl@pop@language{%
3050   \expandafter\bbl@pop@lang\bbl@language@stack-\bbl@language@stack
3051   \let\bbl@ifrestoring\@firstoftwo
3052   \expandafter\bbl@set@language\expandafter{\language}%
3053   \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.

```
3054 \chardef\localeid\z@
3055 \def\bbl@id@last{0} % No real need for a new counter
3056 \def\bbl@id@assign{%
3057   \bbl@ifunset{bbl@id@\language}%
3058   {\count@bbl@id@last\relax
3059    \advance\count@@ne
3060    \bbl@csarg\chardef{id@\language}\count@
3061    \edef\bbl@id@last{\the\count@}%
3062    \ifcase\bbl@engine\or
3063      \directlua{
3064        Babel = Babel or {}
3065        Babel.locale_props = Babel.locale_props or {}
3066        Babel.locale_props[\bbl@id@last] = {}
3067        Babel.locale_props[\bbl@id@last].name = '\language'
3068      }%
3069    \fi}%
3070   {}}%
3071   \chardef\localeid\bbl@c{id@}}
```

The unprotected part of `\selectlanguage`.

```
3072 \expandafter\def\csname selectlanguage \endcsname#1{%
3073   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw\fi
3074   \bbl@push@language
3075   \aftergroup\bbl@pop@language
3076   \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.

```

3077 \def\BabelContentsFiles{toc,lof,lot}
3078 \def\bbl@set@language#1{% from selectlanguage, pop@
3079   \edef\language{%
3080     \ifnum\escapechar=\expandafter`\string#1\@empty
3081     \else\string#1\@empty\fi}%
3082   \select@language{\language}%
3083   % write to auxs
3084   \expandafter\ifx\csname date\language\endcsname\relax\else
3085     \if@files
3086       \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
3087         \protected@write\@auxout{}\string\babel@aux{\language}\fi}%
3088       \fi
3089       \bbl@usehooks{write}\fi}%
3090   \fi
3091 \fi}
3092 \def\select@language#1{% from set@, babel@aux
3093   % set hmap
3094   \ifnum\bbl@hmapsel=\@cclv\chardef\bbl@hmapsel4\relax\fi
3095   % set name
3096   \edef\language{#1}%
3097   \bbl@fixname\language
3098   \expandafter\ifx\csname date\language\endcsname\relax
3099     \IfFileExists{babel-\language.tex}%
3100     {\babelprovide{\language}}%
3101     {}%
3102   \fi
3103   \bbl@iflanguage\language{%
3104     \expandafter\ifx\csname date\language\endcsname\relax
3105       \bbl@error
3106       {Unknown language `#1'. Either you have\\%
3107         misspelled its name, it has not been installed,\\%
3108         or you requested it in a previous run. Fix its name,\\%
3109         install it or just rerun the file, respectively. In\\%
3110         some cases, you may need to remove the aux file}%
3111       {You may proceed, but expect wrong results}%
3112     \else
3113       % set type
3114       \let\bbl@select@type\z@
3115       \expandafter\bbl@switch\expandafter{\language}%
3116     \fi}}
3117 \def\babel@aux#1#2{%
3118   \select@language{#1}%
3119   \bbl@foreach\BabelContentsFiles{%
3120     \@writefile{##1}{\babel@toc{#1}{#2}}}% % TODO - ok in plain?
3121 \def\babel@toc#1#2{%
3122   \select@language{#1}}

```

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

```

3123 \let\select@language\select@language

```

First, check if the user asks for a known language. If so, update the value of \language and call \originalTeX to bring T<sub>E</sub>X 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.

```

3124 \newif\ifbbl@usedategroup
3125 \def\bbl@switch#1{% from select@, foreign@
3126 % make sure there is info for the language if so requested
3127 \bbl@ensureinfo{#1}%
3128 % restore
3129 \originalTeX
3130 \expandafter\def\expandafter\originalTeX\expandafter{%
3131 \csname noextras#1\endcsname
3132 \let\originalTeX\@empty
3133 \babel@beginsave}%
3134 \bbl@usehooks{afterreset}{}%
3135 \languageshortands{none}%
3136 % set the locale id
3137 \bbl@id@assign
3138 % switch captions, date
3139 \ifcase\bbl@select@type
3140 \ifhmode
3141 \hskip\z@skip % trick to ignore spaces
3142 \csname captions#1\endcsname\relax
3143 \csname date#1\endcsname\relax
3144 \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
3145 \else
3146 \csname captions#1\endcsname\relax
3147 \csname date#1\endcsname\relax
3148 \fi
3149 \else
3150 \ifbbl@usedategroup % if \foreign... within \<lang>date
3151 \bbl@usedategroupfalse
3152 \ifhmode
3153 \hskip\z@skip % trick to ignore spaces
3154 \csname date#1\endcsname\relax
3155 \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
3156 \else
3157 \csname date#1\endcsname\relax
3158 \fi
3159 \fi
3160 \fi
3161 % switch extras
3162 \bbl@usehooks{beforeextras}{}%
3163 \csname extras#1\endcsname\relax
3164 \bbl@usehooks{afterextras}{}%
3165 % > babel-ensure
3166 % > babel-sh-<short>
3167 % > babel-bidi
3168 % > babel-fontspec
3169 % hyphenation - case mapping

```



```

3170 \ifcase\bb1@opt@hyphenmap\or
3171 \def\BabelLower##1##2{\lccode##1=##2\relax}%
3172 \ifnum\bb1@hymapsel>4\else
3173 \csname\language @bb1@hyphenmap\endcsname
3174 \fi
3175 \chardef\bb1@opt@hyphenmap\z@
3176 \else
3177 \ifnum\bb1@hymapsel>\bb1@opt@hyphenmap\else
3178 \csname\language @bb1@hyphenmap\endcsname
3179 \fi
3180 \fi
3181 \global\let\bb1@hymapsel\@cclv
3182 % hyphenation - patterns
3183 \bb1@patterns{#1}%
3184 % hyphenation - mins
3185 \babel@savevariable\lefthyphenmin
3186 \babel@savevariable\righthyphenmin
3187 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
3188 \set@hyphenmins\tw@\thr@@\relax
3189 \else
3190 \expandafter\expandafter\expandafter\set@hyphenmins
3191 \csname #1hyphenmins\endcsname\relax
3192 \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.

```

3193 \long\def\otherlanguage#1{%
3194 \ifnum\bb1@hymapsel=\@cclv\let\bb1@hymapsel\thr@@\fi
3195 \csname selectlanguage \endcsname{#1}%
3196 \ignorespaces}

```

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

```

3197 \long\def\endotherlanguage{%
3198 \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`.

```

3199 \expandafter\def\csname otherlanguage*\endcsname#1{%
3200 \ifnum\bb1@hymapsel=\@cclv\chardef\bb1@hymapsel4\relax\fi
3201 \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”.

```

3202 \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.

```

3203 \providecommand\bbl@beforeforeign{}
3204 \edef\foreignlanguage{%
3205   \noexpand\protect
3206   \expandafter\noexpand\csname foreignlanguage \endcsname}
3207 \expandafter\def\csname foreignlanguage \endcsname{%
3208   \@ifstar\bbl@foreign@s\bbl@foreign@x}
3209 \def\bbl@foreign@x#1#2{%
3210   \begingroup
3211     \let\BabelText\@firstofone
3212     \bbl@beforeforeign
3213     \foreign@language{#1}%
3214     \bbl@usehooks{foreign}{}%
3215     \BabelText{#2}% Now in horizontal mode!
3216   \endgroup}
3217 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@@par
3218   \begingroup
3219     {\par}%
3220     \let\BabelText\@firstofone
3221     \foreign@language{#1}%
3222     \bbl@usehooks{foreign*}{}%
3223     \bbl@dirparastext
3224     \BabelText{#2}% Still in vertical mode!
3225     {\par}%
3226   \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`.

```

3227 \def\foreign@language#1{%
3228   % set name
3229   \edef\language#1%
3230   \bbl@fixname\language
3231   \expandafter\ifx\csname date\language\endcsname\relax
3232     \IfFileExists{babel-\language.tex}%
3233     {\babelprovide{\language}}%
3234     {}%
3235   \fi
3236   \bbl@iflanguage\language{%
3237     \expandafter\ifx\csname date\language\endcsname\relax
3238       \bbl@warning % TODO - why a warning, not an error?

```

```

3239      {Unknown language `#1'. Either you have\\%
3240      misspelled its name, it has not been installed,\\%
3241      or you requested it in a previous run. Fix its name,\\%
3242      install it or just rerun the file, respectively. In\\%
3243      some cases, you may need to remove the aux file.\\%
3244      I'll proceed, but expect wrong results.\\%
3245      Reported}%
3246  \fi
3247  % set type
3248  \let\bbl@select@type\@ne
3249  \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.

```

3250 \let\bbl@hyphlist\@empty
3251 \let\bbl@hyphenation@ \relax
3252 \let\bbl@pttnlist\@empty
3253 \let\bbl@patterns@ \relax
3254 \let\bbl@hymapsel=\@cc1v
3255 \def\bbl@patterns#1{%
3256   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
3257     \csname l@#1\endcsname
3258     \edef\bbl@tempa{#1}%
3259   \else
3260     \csname l@#1:\f@encoding\endcsname
3261     \edef\bbl@tempa{#1:\f@encoding}%
3262   \fi
3263   \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}%
3264   % > luatex
3265   \@ifundefined{bbl@hyphenation@}{#1}{% Can be \relax!
3266     \begingroup
3267       \bbl@xin@{, \number\language,}{, \bbl@hyphlist}%
3268     \ifin@ \else
3269       \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}%
3270       \hyphenation{%
3271         \bbl@hyphenation@
3272         \@ifundefined{bbl@hyphenation@#1}{%
3273           \@empty
3274           {\space\csname bbl@hyphenation@#1\endcsname}}%
3275       \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
3276     \fi
3277   \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*`.

```

3278 \def\hyphenrules#1{%
3279   \edef\bbl@tempf{#1}%
3280   \bbl@fixname\bbl@tempf

```

```

3281 \bbl@iflanguage\bbl@tempf{%
3282   \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
3283   \languageshorthands{none}%
3284   \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
3285     \set@hyphenmins\tw@\thr@\relax
3286   \else
3287     \expandafter\expandafter\expandafter\set@hyphenmins
3288     \csname\bbl@tempf hyphenmins\endcsname\relax
3289   \fi}}
3290 \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.

```

3291 \def\providehyphenmins#1#2{%
3292   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
3293     \@namedef{#1hyphenmins}{#2}%
3294   \fi}

```

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

```

3295 \def\set@hyphenmins#1#2{%
3296   \lefthyphenmin#1\relax
3297   \righthyphenmin#2\relax}

```

`\ProvidesLanguage` The identification code for each file is something that was introduced in L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>. 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.

```

3298 \ifx\ProvidesFile\@undefined
3299   \def\ProvidesLanguage#1[#2 #3 #4]{%
3300     \wlog{Language: #1 #4 #3 <#2>}%
3301   }
3302 \else
3303   \def\ProvidesLanguage#1{%
3304     \begingroup
3305       \catcode`\ 10 %
3306       \@makeother\/%
3307       \@ifnextchar[%]
3308         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
3309   \def\@provideslanguage#1[#2]{%
3310     \wlog{Language: #1 #2}%
3311     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
3312   \endgroup}
3313 \fi

```

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

```

3314 \def\LdfInit{%
3315   \chardef\atcatcode=\catcode`\@
3316   \catcode`\@=11\relax
3317   \input babel.def\relax
3318   \catcode`\@=\atcatcode \let\atcatcode\relax
3319   \LdfInit}

```

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

```
3320 \ifx\originalTeX\undefined\let\originalTeX\@empty\fi
```

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

```
3321 \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’:

```
3322 \providecommand\setlocale{%
3323   \bbl@error
3324   {Not yet available}%
3325   {Find an armchair, sit down and wait}}
3326 \let\uselocale\setlocale
3327 \let\locale\setlocale
3328 \let\selectlocale\setlocale
3329 \let\localename\setlocale
3330 \let\textlocale\setlocale
3331 \let\textlanguage\setlocale
3332 \let\languagetext\setlocale
```

## 12.2 Errors

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

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

```
3333 \edef\bbl@nulllanguage{\string\language=0}
3334 \ifx\PackageError\undefined
3335   \def\bbl@error#1#2{%
3336     \begingroup
3337       \newlinechar=`^^J
3338       \def\{^^J(babel) }%
3339       \errhelp{#2}\errmessage{\{#1}%
3340     \endgroup}
3341   \def\bbl@warning#1{%
3342     \begingroup
3343       \newlinechar=`^^J
3344       \def\{^^J(babel) }%
3345       \message{\{#1}%
3346     \endgroup}
3347   \let\bbl@infowarn\bbl@warning
3348   \def\bbl@info#1{%
3349     \begingroup
3350       \newlinechar=`^^J
3351       \def\{^^J}%
```

```

3352     \wlog{#1}%
3353   \endgroup}
3354 \else
3355   \def\bbl@error#1#2{%
3356     \begingroup
3357       \def\{\MessageBreak}%
3358       \PackageError{babel}{#1}{#2}%
3359     \endgroup}
3360   \def\bbl@warning#1{%
3361     \begingroup
3362       \def\{\MessageBreak}%
3363       \PackageWarning{babel}{#1}%
3364     \endgroup}
3365   \def\bbl@infowarn#1{%
3366     \begingroup
3367       \def\{\MessageBreak}%
3368       \GenericWarning
3369         {(babel) \@spaces\@spaces\@spaces}%
3370         {Package babel Info: #1}%
3371     \endgroup}
3372   \def\bbl@info#1{%
3373     \begingroup
3374       \def\{\MessageBreak}%
3375       \PackageInfo{babel}{#1}%
3376     \endgroup}
3377 \fi
3378 \@ifpackagewith{babel}{silent}
3379   {\let\bbl@info@gobble
3380    \let\bbl@infowarn@gobble
3381    \let\bbl@warning@gobble}
3382 {}
3383 \def\bbl@nocaption{\protect\bbl@nocaption@i}
3384 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
3385   \global\@namedef{#2}{\textbf{?#1?}}%
3386   \@nameuse{#2}%
3387   \bbl@warning{%
3388     \@backslashchar#2 not set. Please, define\%
3389     it in the preamble with something like:\%
3390     \string\renewcommand\@backslashchar#2{..}\%
3391     Reported}}
3392 \def\bbl@tentative{\protect\bbl@tentative@i}
3393 \def\bbl@tentative@i#1{%
3394   \bbl@warning{%
3395     Some functions for '#1' are tentative.\%
3396     They might not work as expected and their behavior\%
3397     could change in the future.\%
3398     Reported}}
3399 \def\@nolanerr#1{%
3400   \bbl@error
3401     {You haven't defined the language #1\space yet.\%
3402     Perhaps you misspelled it or your installation\%
3403     is not complete}%
3404     {Your command will be ignored, type <return> to proceed}}
3405 \def\@nopatterns#1{%
3406   \bbl@warning
3407     {No hyphenation patterns were preloaded for\%
3408     the language '#1' into the format.\%
3409     Please, configure your TeX system to add them and\%
3410     rebuild the format. Now I will use the patterns\%

```

```

3411     preloaded for \bbl@nulllanguage\space instead}}
3412 \let\bbl@usehooks\@gobbletwo
3413 </kernel>
3414 <*patterns>

```

## 13 Loading hyphenation patterns

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

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

```

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

```

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

There are some problems with this approach though.

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

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

To make sure that  $\text{\texttt{L\TeX}}$  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.

```

3415 <<Make sure ProvidesFile is defined>>
3416 \ProvidesFile{hyphen.cfg}[\<date>] [\<version>] Babel hyphens]
3417 \xdef\bbl@format{\jobname}
3418 \ifx\AtBeginDocument\@undefined
3419   \def\@empty{}
3420   \let\orig@dump\dump
3421   \def\dump{%
3422     \ifx\@ztryfc\@undefined
3423     \else
3424       \toks0=\expandafter{\@preamblecmds}%
3425       \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
3426       \def\@begindocumenthook{}%
3427     \fi
3428     \let\dump\orig@dump\let\orig@dump\@undefined\dump}

```

```

3429 \fi
3430 <<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.

3431 \def\process@line#1#2 #3 #4 {%
3432   \ifx=#1%
3433     \process@synonym{#2}%
3434   \else
3435     \process@language{#1#2}{#3}{#4}%
3436   \fi
3437   \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.

3438 \toks@{}
3439 \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.

3440 \def\process@synonym#1{%
3441   \ifnum\last@language=\m@ne
3442     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
3443   \else
3444     \expandafter\chardef\csname l@#1\endcsname\last@language
3445     \wlog{\string\l@#1=\string\language\the\last@language}%
3446     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
3447       \csname\language\hyphenmins\endcsname
3448     \let\bbl@elt\relax
3449     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}}%
3450   \fi}

\process@language The macro \process@language is used to process a non-empty line from the ‘configuration
file’. It has three arguments, each delimited by white space. The first argument is the
‘name’ of a language; the second is the name of the file that contains the patterns. The
optional third argument is the name of a file containing hyphenation exceptions.
The first thing to do is call \addlanguage to allocate a pattern register and to make that
register ‘active’. Then the pattern file is read.
For some hyphenation patterns it is needed to load them with a specific font encoding
selected. This can be specified in the file language.dat by adding for instance ‘:T1’ to the
name of the language. The macro \bbl@get@enc extracts the font encoding from the
language name and stores it in \bbl@hyph@enc. The latter can be used in hyphenation files
if you need to set a behavior depending on the given encoding (it is set to empty if no
encoding is given).
Pattern files may contain assignments to \lefthyphenmin and \righthyphenmin. TEX does
not keep track of these assignments. Therefore we try to detect such assignments and store
them in the \<lang>hyphenmins macro. When no assignments were made we provide a
default setting.
Some pattern files contain changes to the \lccode 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.

```

3451 \def\process@language#1#2#3{%
3452   \expandafter\addlanguage\csname l@#1\endcsname
3453   \expandafter\language\csname l@#1\endcsname
3454   \edef\language#1{%
3455     \bbl@hook@everylanguage{#1}%
3456     % > luatex
3457     \bbl@get@enc#1::@@@
3458   \begingroup
3459     \lefthyphenmin\m@ne
3460     \bbl@hook@loadpatterns{#2}%
3461     % > luatex
3462     \ifnum\lefthyphenmin=\m@ne
3463       \else
3464         \expandafter\xdef\csname #1hyphenmins\endcsname{%
3465           \the\lefthyphenmin\the\righthyphenmin}%
3466       \fi
3467   \endgroup
3468   \def\bbl@tempa{#3}%
3469   \ifx\bbl@tempa\@empty\else
3470     \bbl@hook@loadexceptions{#3}%
3471     % > luatex
3472   \fi
3473   \let\bbl@elt\relax
3474   \edef\bbl@languages{%
3475     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
3476   \ifnum\the\language=\z@
3477     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
3478       \set@hyphenmins\tw@\thr@@\relax
3479     \else
3480       \expandafter\expandafter\expandafter\set@hyphenmins
3481       \csname #1hyphenmins\endcsname
3482     \fi
3483     \the\toks@
3484     \toks@{}%
3485   \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.

```

3486 \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.

```

3487 \def\bbl@hook@everylanguage#1{}
3488 \def\bbl@hook@loadpatterns#1{\input #1\relax}
3489 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
3490 \let\bbl@hook@loadkernel\bbl@hook@loadpatterns

```

```

3491 \begingroup
3492 \def\AddBabelHook#1#2{%
3493   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
3494     \def\next{\toks1}%
3495   \else
3496     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
3497   \fi
3498   \next}
3499 \ifx\directlua\undefined
3500   \ifx\XeTeXinputencoding\undefined\else
3501     \input xebabel.def
3502   \fi
3503 \else
3504   \input luababel.def
3505 \fi
3506 \openin1 = babel-\bbl@format.cfg
3507 \ifeof1
3508 \else
3509   \input babel-\bbl@format.cfg\relax
3510 \fi
3511 \closein1
3512 \endgroup
3513 \bbl@hook@loadkernel{switch.def}

```

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

```

3514 \openin1 = language.dat

```

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

```

3515 \def\language{english}%
3516 \ifeof1
3517   \message{I couldn't find the file language.dat,\space
3518     I will try the file hyphen.tex}
3519   \input hyphen.tex\relax
3520   \chardef\l@english\z@
3521 \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.

```

3522 \last@language\m@ne

```

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

```

3523 \loop

```

While reading from the input, it is useful to switch off recognition of the end-of-line character. This saves us stripping off spaces from the contents of the control sequence.

```

3524 \endlinechar\m@ne
3525 \read1 to \bbl@line
3526 \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.

```

3527 \if T\ifeof1F\fi T\relax
3528 \ifx\bbl@line\empty\else
3529   \edef\bbl@line{\bbl@line\space\space\space}%

```

```

3530      \expandafter\process@line\bbl@line\relax
3531      \fi
3532  \repeat

```

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

```

3533  \begingroup
3534      \def\bbl@elt#1#2#3#4{%
3535          \global\language=#2\relax
3536          \gdef\language#1}%
3537      \def\bbl@elt##1##2##3##4{}}%
3538      \bbl@languages
3539  \endgroup
3540 \fi

```

and close the configuration file.

```

3541 \closein1

```

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

```

3542 \if/\the\toks@/\else
3543     \errhelp{language.dat loads no language, only synonyms}
3544     \errmessage{Orphan language synonym}
3545 \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.

```

3546 \let\bbl@line\@undefined
3547 \let\process@line\@undefined
3548 \let\process@synonym\@undefined
3549 \let\process@language\@undefined
3550 \let\bbl@get@enc\@undefined
3551 \let\bbl@hyph@enc\@undefined
3552 \let\bbl@tempa\@undefined
3553 \let\bbl@hook@loadkernel\@undefined
3554 \let\bbl@hook@everylanguage\@undefined
3555 \let\bbl@hook@loadpatterns\@undefined
3556 \let\bbl@hook@loadexceptions\@undefined
3557 </patterns>

```

Here the code for `iniTeX` ends.

## 14 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].

```

3558 <(*More package options)> \equiv
3559 \ifodd\bbl@engine
3560     \DeclareOption{bidi=basic-r}%
3561     {\ExecuteOptions{bidi=basic}}
3562     \DeclareOption{bidi=basic}%
3563     {\let\bbl@beforeforeign\leavevmode
3564      % TODO - to locale_props, not as separate attribute
3565      \newattribute\bbl@attr@dir
3566      % I don't like it, hackish:

```

```

3567 \frozen@everymath\expandafter{%
3568 \expandafter\bb1@mathboxdir\the\frozen@everymath}%
3569 \frozen@everydisplay\expandafter{%
3570 \expandafter\bb1@mathboxdir\the\frozen@everydisplay}%
3571 \bb1@exp{\output{\bodydir\pagedir\the\output}}%
3572 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}}
3573 \else
3574 \DeclareOption{bidi=basic-r}%
3575 {\ExecuteOptions{bidi=basic}}
3576 \DeclareOption{bidi=basic}%
3577 {\bb1@error
3578 {The bidi method `basic' is available only in\\%
3579 luatex. I'll continue with `bidi=default', so\\%
3580 expect wrong results}%
3581 {See the manual for further details.}%
3582 \let\bb1@beforeforeign\leavevmode
3583 \AtEndOfPackage{%
3584 \EnableBabelHook{babel-bidi}%
3585 \bb1@xebidipar}}
3586 \def\bb1@loadxebidi#1{%
3587 \ifx\RTLfootnotetext\@undefined
3588 \AtEndOfPackage{%
3589 \EnableBabelHook{babel-bidi}%
3590 \ifx\fontspec\@undefined
3591 \usepackage{fontspec}% bidi needs fontspec
3592 \fi
3593 \usepackage#1{bidi}}%
3594 \fi}
3595 \DeclareOption{bidi=bidi}%
3596 {\bb1@tentative{bidi=bidi}%
3597 \bb1@loadxebidi}}
3598 \DeclareOption{bidi=bidi-r}%
3599 {\bb1@tentative{bidi=bidi-r}%
3600 \bb1@loadxebidi{[rldocument]}}
3601 \DeclareOption{bidi=bidi-l}%
3602 {\bb1@tentative{bidi=bidi-l}%
3603 \bb1@loadxebidi}}
3604 \fi
3605 \DeclareOption{bidi=default}%
3606 {\let\bb1@beforeforeign\leavevmode
3607 \ifodd\bb1@engine
3608 \newattribute\bb1@attr@dir
3609 \bb1@exp{\output{\bodydir\pagedir\the\output}}%
3610 \fi
3611 \AtEndOfPackage{%
3612 \EnableBabelHook{babel-bidi}%
3613 \ifodd\bb1@engine\else
3614 \bb1@xebidipar
3615 \fi}}
3616 <</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. `\bb1@font` replaces hardcoded font names inside `\. .family` by the corresponding macro `\. .default`.

```

3617 <<(*Font selection)>> ≡
3618 \bb1@trace{Font handling with fontspec}
3619 \@onlypreamble\babelfont
3620 \newcommand\babelfont[2][]{% 1=langs/scripts 2=fam
3621 \bb1@foreach{#1}{%

```

```

3622 \expandafter\ifx\csname date##1\endcsname\relax
3623 \IfFileExists{babel-##1.tex}%
3624 {\babelprovide{##1}}%
3625 {}%
3626 \fi}%
3627 \edef\bbl@tempa{#1}%
3628 \def\bbl@tempb{#2}% Used by \bbl@bblfont
3629 \ifx\fontspec\undefined
3630 \usepackage{fontspec}%
3631 \fi
3632 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
3633 \bbl@bblfont}
3634 \newcommand\bbl@bblfont[2][{}% 1=features 2=fontname, @font=rm|sf|tt
3635 \bbl@ifunset{\bbl@tempb family}%
3636 {\bbl@providefam{\bbl@tempb}}%
3637 {\bbl@exp{%
3638 \\\bbl@sreplace\<\bbl@tempb family >%
3639 {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
3640 % For the default font, just in case:
3641 \bbl@ifunset{\bbl@lsys\language}{\bbl@provide@lsys{\language}}}%
3642 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
3643 {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
3644 \bbl@exp{%
3645 \let\<\bbl@\bbl@tempb dflt@\language>\<\bbl@\bbl@tempb dflt@>%
3646 \\\bbl@font@set\<\bbl@\bbl@tempb dflt@\language>%
3647 \<\bbl@tempb default>\<\bbl@tempb family>}}%
3648 {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *srt
3649 \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:

```

3650 \def\bbl@providefam#1{%
3651 \bbl@exp{%
3652 \\\newcommand\<#1default>{}% Just define it
3653 \\\bbl@add@list\\bbl@font@fams{#1}%
3654 \\\DeclareRobustCommand\<#1family>{%
3655 \\\not@math@alphabet\<#1family>\relax
3656 \\\fontfamily\<#1default>\selectfont}%
3657 \\\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.

```

3658 \def\bbl@nostdfont#1{%
3659 \bbl@ifunset{\bbl@WFF@\f@family}%
3660 {\bbl@csarg\gdef{WFF@\f@family}}}% Flag, to avoid dupl warns
3661 \bbl@infowarn{The current font is not a babel standard family:\%
3662 #1%
3663 \fontname\font\\%
3664 There is nothing intrinsically wrong with this warning, and\\%
3665 you can ignore it altogether if you do not need these\\%
3666 families. But if they are used in the document, you should be\\%
3667 aware 'babel' will no set Script and Language for them, so\\%
3668 you may consider defining a new family with \string\babelfont.\\%
3669 See the manual for further details about \string\babelfont.\\%
3670 Reported}}
3671 {}}%
3672 \gdef\bbl@switchfont{%
3673 \bbl@ifunset{\bbl@lsys\language}{\bbl@provide@lsys{\language}}}%
3674 \bbl@exp{% eg Arabic -> arabic
3675 \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}}%

```

```

3676 \bbl@foreach\bbl@font@fams{%
3677   \bbl@ifunset\bbl@##1dflt@\language}% (1) language?
3678   {\bbl@ifunset\bbl@##1dflt@*\bbl@tempa}% (2) from script?
3679   {\bbl@ifunset\bbl@##1dflt@}% 2=F - (3) from generic?
3680   {}% 123=F - nothing!
3681   {\bbl@exp{% 3=T - from generic
3682     \global\let<\bbl@##1dflt@\language>%
3683     \<\bbl@##1dflt@>}}}%
3684   {\bbl@exp{% 2=T - from script
3685     \global\let<\bbl@##1dflt@\language>%
3686     \<\bbl@##1dflt@*\bbl@tempa>}}}%
3687   {}}% 1=T - language, already defined
3688 \def\bbl@tempa{\bbl@nostdfont{}}%
3689 \bbl@foreach\bbl@font@fams{% don't gather with prev for
3690   \bbl@ifunset\bbl@##1dflt@\language}%
3691   {\bbl@cs{famrst@##1}%
3692     \global\bbl@csarg\let{famrst@##1}\relax}%
3693   {\bbl@exp{% order is relevant
3694     \\bbl@add\\originalTeX{%
3695       \\bbl@font@rst{\bbl@cl{##1dflt}}%
3696       \<##1default>\<##1family>{##1}}%
3697     \\bbl@font@set<\bbl@##1dflt@\language>% the main part!
3698     \<##1default>\<##1family>}}}%
3699 \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.

```

3700 \ifx\family\undefined\else % if latex
3701   \ifcase\bbl@engine % if pdftex
3702     \let\bbl@cckstdfont\relax
3703   \else
3704     \def\bbl@cckstdfont{%
3705       \begingroup
3706       \global\let\bbl@cckstdfont\relax
3707       \let\bbl@tempa\@empty
3708       \bbl@foreach\bbl@font@fams{%
3709         \bbl@ifunset\bbl@##1dflt@}%
3710         {\nameuse{##1family}%
3711           \bbl@csarg\gdef{WFF@\family}}}% Flag
3712         \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= \family\\}%
3713           \space\space\fontname\font\\}%
3714         \bbl@csarg\xdef{##1dflt@}{\family}%
3715         \expandafter\xdef\csname ##1default\endcsname{\family}}%
3716       {}}%
3717     \ifx\bbl@tempa\@empty\else
3718       \bbl@infowarn{The following font families will use the default\\%
3719         settings for all languages:\\%
3720         \bbl@tempa
3721         There is nothing intrinsically wrong with it, but\\%
3722         'babel' will no set Script and Language, which could\\%
3723         be relevant in some languages. If your document uses\\%
3724         these families, consider redefining them with \string\babelfont.\\%
3725         Reported}%
3726     \fi
3727   \endgroup}
3728 \fi
3729 \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.

```

3730 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
3731 \bbl@xin@{<>}{#1}%
3732 \ifin@
3733 \bbl@exp{\bbl@fontspec@set\#1\expandafter\@gobbletwo#1\#3}%
3734 \fi
3735 \bbl@exp{%
3736 \def\#2{#1}% eg, \rmdefault{\bbl@rmdflt@lang}
3737 \bbl@ifsamestring{#2}{\f@family}{\#3\let\bbl@tempa\relax}{}}
3738 % TODO - next should be global?, but even local does its job. I'm
3739 % still not sure -- must investigate:
3740 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
3741 \let\bbl@tempe\bbl@mapselect
3742 \let\bbl@mapselect\relax
3743 \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
3744 \let#4\@empty % Make sure \renewfontfamily is valid
3745 \bbl@exp{%
3746 \let\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
3747 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
3748 {\bbl@newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
3749 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
3750 {\bbl@newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
3751 \bbl@renewfontfamily\#4%
3752 [\bbl@cs{lsys@languagename},#2]{#3}% ie \bbl@exp{.}\#3}
3753 \begingroup
3754 #4%
3755 \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
3756 \endgroup
3757 \let#4\bbl@temp@fam
3758 \bbl@exp{\let<\bbl@stripslash#4\space>\bbl@temp@pfam
3759 \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.

```

3760 \def\bbl@font@rst#1#2#3#4{%
3761 \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.

```

3762 \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 :-).

```

3763 \newcommand\babelFSstore[2][{}]{%
3764 \bbl@ifblank{#1}%
3765 {\bbl@csarg\def{sname@#2}{Latin}}%
3766 {\bbl@csarg\def{sname@#2}{#1}}%
3767 \bbl@provide@dirs{#2}%
3768 \bbl@csarg\ifnum{wdir@#2}>\z@
3769 \let\bbl@beforeforeign\leavevmode
3770 \EnableBabelHook{babel-bidi}%
3771 \fi
3772 \bbl@foreach{#2}{%
3773 \bbl@FSstore{##1}{rm}\rmdefault\bbl@save\rmdefault

```

```

3774 \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
3775 \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
3776 \def\bbl@FSstore#1#2#3#4{%
3777 \bbl@csarg\edef{#2default#1}{#3}%
3778 \expandafter\addto\csname extras#1\endcsname{%
3779 \let#4#3%
3780 \ifx#3\f@family
3781 \edef#3{\csname bbl@#2default#1\endcsname}%
3782 \fontfamily{#3}\selectfont
3783 \else
3784 \edef#3{\csname bbl@#2default#1\endcsname}%
3785 \fi}%
3786 \expandafter\addto\csname noextras#1\endcsname{%
3787 \ifx#3\f@family
3788 \fontfamily{#4}\selectfont
3789 \fi
3790 \let#3#4}}
3791 \let\bbl@langfeatures\@empty
3792 \def\babelFSfeatures{% make sure \fontspec is redefined once
3793 \let\bbl@ori@fontspec\fontspec
3794 \renewcommand\fontspec[1][{}]{%
3795 \bbl@ori@fontspec[\bbl@langfeatures##1]}
3796 \let\babelFSfeatures\bbl@FSfeatures
3797 \babelFSfeatures}
3798 \def\bbl@FSfeatures#1#2{%
3799 \expandafter\addto\csname extras#1\endcsname{%
3800 \babel@save\bbl@langfeatures
3801 \edef\bbl@langfeatures{#2,}}
3802 <</Font selection>>

```

## 15 Hooks for XeTeX and LuaTeX

### 15.1 XeTeX

Unfortunately, the current encoding cannot be retrieved and therefore it is reset always to `utf8`, which seems a sensible default.

$\LaTeX$  sets many “codes” just before loading `hyphen.cfg`. That is not a problem in `luatex`, but in `xetex` they must be reset to the proper value. Most of the work is done in `xe(la)tex.ini`, so here we just “undo” some of the changes done by  $\LaTeX$ . Anyway, for consistency `LuaTeX` also resets the catcodes.

```

3803 <<(*Restore Unicode catcodes before loading patterns)>> ≡
3804 \begingroup
3805 % Reset chars "80-"C0 to category "other", no case mapping:
3806 \catcode`\@=11 \count@=128
3807 \loop\ifnum\count@<192
3808 \global\uccode\count@=0 \global\lccode\count@=0
3809 \global\catcode\count@=12 \global\sfcodes\count@=1000
3810 \advance\count@ by 1 \repeat
3811 % Other:
3812 \def\O ##1 {%
3813 \global\uccode"##1=0 \global\lccode"##1=0
3814 \global\catcode"##1=12 \global\sfcodes"##1=1000 }%
3815 % Letter:
3816 \def\L ##1 ##2 ##3 {\global\catcode"##1=11
3817 \global\uccode"##1="##2
3818 \global\lccode"##1="##3
3819 % Uppercase letters have sfcodes=999:

```



```

3820     \ifnum"##1="##3 \else \global\sfcodes"##1=999 \fi }%
3821     % Letter without case mappings:
3822     \def\l ##1 {\L ##1 ##1 ##1 }%
3823     \l 00AA
3824     \L 00B5 039C 00B5
3825     \l 00BA
3826     \O 00D7
3827     \l 00DF
3828     \O 00F7
3829     \L 00FF 0178 00FF
3830 \endgroup
3831 \input #1\relax
3832 <</Restore Unicode catcodes before loading patterns>>

```

Some more common code.

```

3833 <<Footnote changes>> ≡
3834 \bbl@trace{Bidi footnotes}
3835 \ifx\bbl@beforeforeign\leavevmode
3836 \def\bbl@footnote#1#2#3{%
3837   \@ifnextchar[%
3838     {\bbl@footnote@o{#1}{#2}{#3}}%
3839     {\bbl@footnote@x{#1}{#2}{#3}}}
3840 \def\bbl@footnote@x#1#2#3#4{%
3841   \bgroup
3842   \select@language@x{\bbl@main@language}%
3843   \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
3844   \egroup}
3845 \def\bbl@footnote@o#1#2#3[#4]#5{%
3846   \bgroup
3847   \select@language@x{\bbl@main@language}%
3848   \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
3849   \egroup}
3850 \def\bbl@footnotetext#1#2#3{%
3851   \@ifnextchar[%
3852     {\bbl@footnotetext@o{#1}{#2}{#3}}%
3853     {\bbl@footnotetext@x{#1}{#2}{#3}}}
3854 \def\bbl@footnotetext@x#1#2#3#4{%
3855   \bgroup
3856   \select@language@x{\bbl@main@language}%
3857   \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
3858   \egroup}
3859 \def\bbl@footnotetext@o#1#2#3[#4]#5{%
3860   \bgroup
3861   \select@language@x{\bbl@main@language}%
3862   \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
3863   \egroup}
3864 \def\BabelFootnote#1#2#3#4{%
3865   \ifx\bbl@fn@footnote\undefined
3866     \let\bbl@fn@footnote\footnote
3867   \fi
3868   \ifx\bbl@fn@footnotetext\undefined
3869     \let\bbl@fn@footnotetext\footnotetext
3870   \fi
3871   \bbl@ifblank{#2}%
3872     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
3873     \@namedef{\bbl@stripslash#1text}%
3874       {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
3875     {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
3876     \@namedef{\bbl@stripslash#1text}%

```

```

3877      {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}{#3}{#4}}}}
3878 \fi
3879 <</Footnote changes>>

```

Now, the code.

```

3880 (*xetex)
3881 \def\BabelStringsDefault{unicode}
3882 \let\xebbl@stop\relax
3883 \AddBabelHook{xetex}{encodedcommands}{%
3884   \def\bbl@tempa{#1}%
3885   \ifx\bbl@tempa\@empty
3886     \XeTeXinputencoding"bytes"%
3887   \else
3888     \XeTeXinputencoding"#1"%
3889   \fi
3890   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
3891 \AddBabelHook{xetex}{stopcommands}{%
3892   \xebbl@stop
3893   \let\xebbl@stop\relax}
3894 \def\bbl@intraspace#1 #2 #3\@@{%
3895   \bbl@csarg\gdef{\xeisp@\language\language}%
3896   {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
3897 \def\bbl@intrapenalty#1\@@{%
3898   \bbl@csarg\gdef{\xeipn@\language\language}%
3899   {\XeTeXlinebreakpenalty #1\relax}}
3900 \def\bbl@provide@intraspace{%
3901   \bbl@xin@{\bbl@cl{lnbrk}}{s}%
3902   \ifin@else\bbl@xin@{\bbl@cl{lnbrk}}{c}\fi
3903   \ifin@
3904     \bbl@ifunset{\bbl@intsp@\language\language}{%
3905       {\expandafter\ifx\csname\bbl@intsp@\language\language\endcsname\@empty\else
3906         \ifx\bbl@KVP@intraspace\@nil
3907           \bbl@exp{%
3908             \bbl@intraspace\bbl@cl{intsp}\@@}%
3909         \fi
3910         \ifx\bbl@KVP@intrapenalty\@nil
3911           \bbl@intrapenalty0\@@
3912         \fi
3913       \fi
3914       \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
3915         \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
3916       \fi
3917       \ifx\bbl@KVP@intrapenalty\@nil\else
3918         \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
3919       \fi
3920       \bbl@exp{%
3921         \bbl@add\<extras\language>{%
3922           \XeTeXlinebreaklocale "\bbl@cl{lbcpr}"%
3923           \<bbl@xeisp@\language\language>%
3924           \<bbl@xeipn@\language\language>%
3925           \bbl@toglobal\<extras\language>%
3926           \bbl@add\<noextras\language>{%
3927             \XeTeXlinebreaklocale "en"%
3928             \bbl@toglobal\<noextras\language>}}%
3929       \ifx\bbl@ispacesize\undefined
3930         \gdef\bbl@ispacesize{\bbl@cl{\xeisp}}%
3931       \ifx\AtBeginDocument\@notprerr
3932         \expandafter\@secondoftwo % to execute right now
3933       \fi

```

```

3934 \AtBeginDocument{%
3935 \expandafter\bb1@add
3936 \csname selectfont \endcsname{\bb1@ispace size}%
3937 \expandafter\bb1@tglobal\csname selectfont \endcsname}%
3938 \fi}%
3939 \fi}
3940 \AddBabelHook{xetex}{loadkernel}{%
3941 <<Restore Unicode catcodes before loading patterns>>}}
3942 \ifx\DisableBabelHook\@undefined\endinput\fi
3943 \AddBabelHook{babel-fontspec}{afterextras}{\bb1@switchfont}
3944 \AddBabelHook{babel-fontspec}{beforestart}{\bb1@cckstdfonts}
3945 \DisableBabelHook{babel-fontspec}
3946 <<Font selection>>
3947 \input txtbabel.def
3948 </xetex>

```

## 15.2 Layout

*In progress.*

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titleps, and geometry.

\bb1@startskip and \bb1@endskip are available to package authors. Thanks to the T<sub>E</sub>X expansion mechanism the following constructs are valid: \adim\bb1@startskip, \advance\bb1@startskip\adim, \bb1@startskip\adim.

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

```

3949 < *texet >
3950 \providecommand\bb1@provide@intraspace{}
3951 \bb1@trace{Redefinitions for bidi layout}
3952 \def\bb1@sspre@caption{%
3953 \bb1@exp{\everyhbox{\bb1@textdir\bb1@cs{wdir@\bb1@main@language}}}}
3954 \ifx\bb1@opt@layout\@nnil\endinput\fi % No layout
3955 \def\bb1@startskip{\ifcase\bb1@thepardir\leftskip\else\rightskip\fi}
3956 \def\bb1@endskip{\ifcase\bb1@thepardir\rightskip\else\leftskip\fi}
3957 \ifx\bb1@beforeforeign\leavevmode % A poor test for bidi=
3958 \def\@hangfrom#1{%
3959 \setbox\@tempboxa\hbox{{#1}}}%
3960 \hangindent\ifcase\bb1@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
3961 \noindent\box\@tempboxa}
3962 \def\raggedright{%
3963 \let\@centercr
3964 \bb1@startskip\z@skip
3965 \@rightskip\@flushglue
3966 \bb1@endskip\@rightskip
3967 \parindent\z@
3968 \parfillskip\bb1@startskip}
3969 \def\raggedleft{%
3970 \let\@centercr
3971 \bb1@startskip\@flushglue
3972 \bb1@endskip\z@skip
3973 \parindent\z@
3974 \parfillskip\bb1@endskip}
3975 \fi
3976 \IfBabelLayout{lists}
3977 {\bb1@sreplace\list
3978 {\@totalleftmargin\leftmargin}{\@totalleftmargin\bb1@listleftmargin}%
3979 \def\bb1@listleftmargin{%

```

```

3980 \ifcase\bb1@thepardir\leftmargin\else\rightmargin\fi}%
3981 \ifcase\bb1@engine
3982 \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
3983 \def\p@enumiii{\p@enumii}\theenumii{}\fi
3984 \fi
3985 \bb1@sreplace\@verbatim
3986 {\leftskip\@totalleftmargin}%
3987 {\bb1@startskip\textwidth
3988 \advance\bb1@startskip-\linewidth}}%
3989 \bb1@sreplace\@verbatim
3990 {\rightskip\z@skip}%
3991 {\bb1@endskip\z@skip}}%
3992 {}
3993 \IfBabelLayout{contents}
3994 {\bb1@sreplace\@dottedtocline{\leftskip}{\bb1@startskip}%
3995 \bb1@sreplace\@dottedtocline{\rightskip}{\bb1@endskip}}
3996 {}
3997 \IfBabelLayout{columns}
3998 {\bb1@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bb1@outputbox}%
3999 \def\bb1@outputbox#1{%
4000 \hb@xt@\textwidth{%
4001 \hskip\columnwidth
4002 \hfil
4003 {\normalcolor\vrule \@width\columnseprule}%
4004 \hfil
4005 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4006 \hskip-\textwidth
4007 \hb@xt@\columnwidth{\box\@outputbox \hss}%
4008 \hskip\columnsep
4009 \hskip\columnwidth}}}%
4010 {}
4011 <<Footnote changes>>
4012 \IfBabelLayout{footnotes}%
4013 {\BabelFootnote\footnote\language\{}}%
4014 \BabelFootnote\localfootnote\language\{}}%
4015 \BabelFootnote\mainfootnote\{}}{}
4016 {}

```

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

```

4017 \IfBabelLayout{counters}%
4018 {\let\bb1@latinarabic=\@arabic
4019 \def\@arabic#1{\babelsublr{\bb1@latinarabic#1}}%
4020 \let\bb1@asciroman=\@roman
4021 \def\@roman#1{\babelsublr{\ensureascii{\bb1@asciroman#1}}}%
4022 \let\bb1@asciiRoman=\@Roman
4023 \def\@Roman#1{\babelsublr{\ensureascii{\bb1@asciiRoman#1}}}}{}
4024 </texxet>

```

### 15.3 LuaTeX

The new loader for luatex is based solely on language.dat, which is read on the fly. The code shouldn't be executed when the format is build, so we check if \AddBabelHook is defined. Then comes a modified version of the loader in hyphen.cfg (without the hyphenmins stuff, which is under the direct control of babel).

The names \l@<language> are defined and take some value from the beginning because all ldf files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first

selected (which usually means when the `ldf` finishes). If a language has been loaded, `\bbl@hyphendata@<num>` exists (with the names of the files read). The default setup preloads the first language into the format. This is intended mainly for ‘english’, so that it’s available without further intervention from the user. To avoid duplicating it, the following rule applies: if the “0th” language and the first language in `language.dat` have the same name then just ignore the latter. If there are new synonymous, they are added, but note if the language patterns have not been preloaded they won’t at run time.

Other preloaded languages could be read twice, if they have been preloaded into the format. This is not optimal, but it shouldn’t happen very often – with `luatex` patterns are best loaded when the document is typeset, and the “0th” language is preloaded just for backwards compatibility.

As of 1.1b, `lua(e)tex` is taken into account. Formerly, loading of patterns on the fly didn’t work in this format, but with the new loader it does. Unfortunately, the format is not based on `babel`, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format `language.dat` is used (under the principle of a single source), instead of `language.def`.

Of course, there is room for improvements, like tools to read and reassign languages, which would require modifying the language list, and better error handling. We need catcode tables, but no format (targeted by `babel`) provide a command to allocate them (although there are packages like `ctablestack`). For the moment, a dangerous approach is used – just allocate a high random number and cross the fingers. To complicate things, `etex.sty` changes the way languages are allocated.

```

4025 (*luatex)
4026 \ifx\AddBabelHook\undefined
4027 \bbl@trace{Read language.dat}
4028 \ifx\bbl@readstream\undefined
4029 \csname newread\endcsname\bbl@readstream
4030 \fi
4031 \begingroup
4032 \toks@{}
4033 \count@ \z@ % 0=start, 1=0th, 2=normal
4034 \def\bbl@process@line#1#2 #3 #4 {%
4035   \ifx=#1%
4036     \bbl@process@synonym{#2}%
4037   \else
4038     \bbl@process@language{#1#2}{#3}{#4}%
4039   \fi
4040   \ignorespaces}
4041 \def\bbl@manylang{%
4042   \ifnum\bbl@last>\@ne
4043     \bbl@info{Non-standard hyphenation setup}%
4044   \fi
4045   \let\bbl@manylang\relax}
4046 \def\bbl@process@language#1#2#3{%
4047   \ifcase\count@
4048     \@ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
4049   \or
4050     \count@\tw@
4051   \fi
4052   \ifnum\count@=\tw@
4053     \expandafter\addlanguage\csname l@#1\endcsname
4054     \language\allocationnumber
4055     \chardef\bbl@last\allocationnumber
4056     \bbl@manylang
4057     \let\bbl@elt\relax

```

```

4058 \xdef\bbl@languages{%
4059 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4060 \fi
4061 \the\toks@
4062 \toks@{}}
4063 \def\bbl@process@synonym@aux#1#2{%
4064 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4065 \let\bbl@elt\relax
4066 \xdef\bbl@languages{%
4067 \bbl@languages\bbl@elt{#1}{#2}{}}}%
4068 \def\bbl@process@synonym#1{%
4069 \ifcase\count@
4070 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4071 \or
4072 \ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}}%
4073 \else
4074 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4075 \fi}
4076 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4077 \chardef\l@english\z@
4078 \chardef\l@USenglish\z@
4079 \chardef\bbl@last\z@
4080 \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
4081 \gdef\bbl@languages{%
4082 \bbl@elt{english}{0}{hyphen.tex}}%
4083 \bbl@elt{USenglish}{0}{}}
4084 \else
4085 \global\let\bbl@languages@format\bbl@languages
4086 \def\bbl@elt#1#2#3#4{% Remove all except language 0
4087 \ifnum#2>\z@\else
4088 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4089 \fi}%
4090 \xdef\bbl@languages{\bbl@languages}%
4091 \fi
4092 \def\bbl@elt#1#2#3#4{\@namedef{zth#1}{}} % Define flags
4093 \bbl@languages
4094 \openin\bbl@readstream=language.dat
4095 \ifeof\bbl@readstream
4096 \bbl@warning{I couldn't find language.dat. No additional\\%
4097 patterns loaded. Reported}%
4098 \else
4099 \loop
4100 \endlinechar\m@ne
4101 \read\bbl@readstream to \bbl@line
4102 \endlinechar\^^M
4103 \if T\ifeof\bbl@readstream F\fi T\relax
4104 \ifx\bbl@line\empty\else
4105 \edef\bbl@line{\bbl@line\space\space\space}%
4106 \expandafter\bbl@process@line\bbl@line\relax
4107 \fi
4108 \repeat
4109 \fi
4110 \endgroup
4111 \bbl@trace{Macros for reading patterns files}
4112 \def\bbl@get@enc#1:#2:#3@@@{\def\bbl@hyph@enc{#2}}
4113 % TODO - Harcoded value:
4114 \ifx\babelcatcodetablenum\@undefined
4115 \def\babelcatcodetablenum{5211}
4116 \fi

```

```

4117 \def\bbl@luapatterns#1#2{%
4118   \bbl@get@enc#1::\@@@
4119   \setbox\z@\hbox\bgroup
4120     \begingroup
4121       \ifx\catcodetable\undefined
4122         \let\savecatcodetable\luatexsavecatcodetable
4123         \let\initcatcodetable\luatexinitcatcodetable
4124         \let\catcodetable\luatexcatcodetable
4125       \fi
4126       \savecatcodetable\babelcatcodetablenum\relax
4127       \initcatcodetable\numexpr\babelcatcodetablenum+1\relax
4128       \catcodetable\numexpr\babelcatcodetablenum+1\relax
4129       \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4130       \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~ =13
4131       \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4132       \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4133       \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4134       \catcode`\'=12 \catcode`\'=12 \catcode`\`=12
4135       \input #1\relax
4136       \catcodetable\babelcatcodetablenum\relax
4137     \endgroup
4138   \def\bbl@tempa{#2}%
4139   \ifx\bbl@tempa\empty\else
4140     \input #2\relax
4141   \fi
4142 \egroup}%
4143 \def\bbl@patterns@lua#1{%
4144   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4145     \csname l@#1\endcsname
4146     \edef\bbl@tempa{#1}%
4147   \else
4148     \csname l@#1:\f@encoding\endcsname
4149     \edef\bbl@tempa{#1:\f@encoding}%
4150   \fi\relax
4151   \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4152   \@ifundefined{bbl@hyphendata@the\language}%
4153   {\def\bbl@elt##1##2##3##4{%
4154     \ifnum##2=\csname l@#1:\f@encoding\endcsname % #2=spanish, dutch:OT1...
4155     \def\bbl@tempb{##3}%
4156     \ifx\bbl@tempb\empty\else % if not a synonymous
4157       \def\bbl@tempc{##3}{##4}%
4158     \fi
4159     \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4160   \fi}%
4161   \bbl@languages
4162   \@ifundefined{bbl@hyphendata@the\language}%
4163   {\bbl@info{No hyphenation patterns were set for\%
4164     language '\bbl@tempa'. Reported}}%
4165   {\expandafter\expandafter\expandafter\bbl@luapatterns
4166     \csname bbl@hyphendata@the\language\endcsname}}}%
4167 \endinput\fi
4168 \begingroup
4169 \catcode`\%=12
4170 \catcode`\'=12
4171 \catcode`\`=12
4172 \catcode`\:=12
4173 \directlua{
4174   Babel = Babel or {}
4175   function Babel.bytes(line)

```

```

4176     return line:gsub("(.)",
4177         function (chr) return unicode.utf8.char(string.byte(chr)) end)
4178 end
4179 function Babel.begin_process_input()
4180     if luatexbase and luatexbase.add_to_callback then
4181         luatexbase.add_to_callback('process_input_buffer',
4182             Babel.bytes, 'Babel.bytes')
4183     else
4184         Babel.callback = callback.find('process_input_buffer')
4185         callback.register('process_input_buffer', Babel.bytes)
4186     end
4187 end
4188 function Babel.end_process_input ()
4189     if luatexbase and luatexbase.remove_from_callback then
4190         luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
4191     else
4192         callback.register('process_input_buffer', Babel.callback)
4193     end
4194 end
4195 function Babel.addpatterns(pp, lg)
4196     local lg = lang.new(lg)
4197     local pats = lang.patterns(lg) or ''
4198     lang.clear_patterns(lg)
4199     for p in pp:gmatch('[^%s]+') do
4200         ss = ''
4201         for i in string.utfcharacters(p:gsub('%d', '')) do
4202             ss = ss .. '%d?' .. i
4203         end
4204         ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
4205         ss = ss:gsub('%.%%d%?$', '%%.')
4206         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4207         if n == 0 then
4208             tex.sprint(
4209                 [[\string\csname\space bbl@info\endcsname{New pattern: }]]
4210                 .. p .. [[{}]])
4211             pats = pats .. ' ' .. p
4212         else
4213             tex.sprint(
4214                 [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
4215                 .. p .. [[{}]])
4216         end
4217     end
4218     lang.patterns(lg, pats)
4219 end
4220 }
4221 \endgroup
4222 \ifx\newattribute\@undefined\else
4223     \newattribute\bbl@attr@locale
4224     \AddBabelHook{luatex}{beforeextras}{%
4225         \setattribute\bbl@attr@locale\localeid}
4226 \fi
4227 \def\BabelStringsDefault{unicode}
4228 \let\luabbl@stop\relax
4229 \AddBabelHook{luatex}{encodedcommands}{%
4230     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
4231     \ifx\bbl@tempa\bbl@tempb\else
4232         \directlua{Babel.begin_process_input()}%
4233         \def\luabbl@stop{%
4234             \directlua{Babel.end_process_input()}}%

```



```

4235 \fi}%
4236 \AddBabelHook{luatex}{stopcommands}{%
4237 \luabbbl@stop
4238 \let\luabbbl@stop\relax}
4239 \AddBabelHook{luatex}{patterns}{%
4240 \@ifundefined{bbl@hyphendata@the\language}%
4241 {\def\bbl@elt##1##2##3##4{%
4242 \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
4243 \def\bbl@tempb{##3}%
4244 \ifx\bbl@tempb@empty\else % if not a synonymous
4245 \def\bbl@tempc{##3}{##4}}%
4246 \fi
4247 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4248 \fi}%
4249 \bbl@languages
4250 \@ifundefined{bbl@hyphendata@the\language}%
4251 {\bbl@info{No hyphenation patterns were set for\%
4252 language '#2'. Reported}}%
4253 {\expandafter\expandafter\expandafter\bbl@luapatterns
4254 \csname bbl@hyphendata@the\language\endcsname}}}%
4255 \@ifundefined{bbl@patterns@}{}%
4256 \begingroup
4257 \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
4258 \ifin\else
4259 \ifx\bbl@patterns@empty\else
4260 \directlua{ Babel.addpatterns(
4261 [[\bbl@patterns@]], \number\language) }%
4262 \fi
4263 \@ifundefined{bbl@patterns@#1}%
4264 \@empty
4265 {\directlua{ Babel.addpatterns(
4266 [[\space\csname bbl@patterns@#1\endcsname]],
4267 \number\language) }}%
4268 \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
4269 \fi
4270 \endgroup}%
4271 \bbl@exp{%
4272 \bbl@ifunset{bbl@prehc@language}{}%
4273 {\bbl@ifblank{\bbl@cs{prehc@language}}}%
4274 {\prehyphenchar=\bbl@c1{prehc}\relax}}}%
4275 \AddBabelHook{luatex}{everylanguage}{%
4276 \def\process@language##1##2##3{%
4277 \def\process@line####1####2 ####3 ####4 {}}%
4278 \AddBabelHook{luatex}{loadpatterns}{%
4279 \input #1\relax
4280 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4281 {#1}{}}%
4282 \AddBabelHook{luatex}{loadexceptions}{%
4283 \input #1\relax
4284 \def\bbl@tempb##1##2{##1}{##1}%
4285 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
4286 {\expandafter\expandafter\expandafter\bbl@tempb
4287 \csname bbl@hyphendata@the\language\endcsname}}

```

`\babelpatterns` This macro adds patterns. Two macros are used to store them: `\bbl@patterns@` for the global ones and `\bbl@patterns@<lang>` for language ones. We make sure there is a space between words when multiple commands are used.

```
4288 \@onlypreamble\babelpatterns
```

```

4289 \AtEndOfPackage{%
4290   \newcommand\babelpatterns[2][\@empty]{%
4291     \ifx\babel@patterns\relax
4292       \let\babel@patterns@\@empty
4293     \fi
4294     \ifx\babel@pttnlist\@empty\else
4295       \babel@warning{%
4296         You must not intermingle \string\selectlanguage\space and\\%
4297         \string\babelpatterns\space or some patterns will not\\%
4298         be taken into account. Reported}%
4299       \fi
4300       \ifx\@empty#1%
4301         \protected@edef\babel@patterns@\babel@patterns@\space#2}%
4302       \else
4303         \edef\babel@tempb{\zap@space#1 \@empty}%
4304         \babel@for\babel@tempa\babel@tempb{%
4305           \babel@fixname\babel@tempa
4306           \babel@iflanguage\babel@tempa{%
4307             \babel@csarg\protected@edef{patterns@\babel@tempa}{%
4308               \@ifundefined{babel@patterns@\babel@tempa}%
4309               \@empty
4310               {\csname babel@patterns@\babel@tempa\endcsname\space}%
4311               #2}}}%
4312       \fi}}

```

## 15.4 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).

```

4313 \directlua{
4314   Babel = Babel or {}
4315   Babel.linebreaking = Babel.linebreaking or {}
4316   Babel.linebreaking.before = {}
4317   Babel.linebreaking.after = {}
4318   Babel.locale = {} % Free to use, indexed with \localeid
4319   function Babel.linebreaking.add_before(func)
4320     tex.print([[noexpand\csname babel@luahyphenate\endcsname]])
4321     table.insert(Babel.linebreaking.before , func)
4322   end
4323   function Babel.linebreaking.add_after(func)
4324     tex.print([[noexpand\csname babel@luahyphenate\endcsname]])
4325     table.insert(Babel.linebreaking.after, func)
4326   end
4327 }
4328 \def\babel@intraspace#1 #2 #3\@{
4329   \directlua{
4330     Babel = Babel or {}
4331     Babel.intraspaces = Babel.intraspaces or {}
4332     Babel.intraspaces['\csname babel@sbcp@\language\endcsname'] = %
4333       {b = #1, p = #2, m = #3}
4334     Babel.locale_props[\the\localeid].intraspace = %
4335       {b = #1, p = #2, m = #3}
4336   }}
4337 \def\babel@intrapenalty#1\@{
4338   \directlua{

```

```

4339 Babel = Babel or {}
4340 Babel.intrapealties = Babel.intrapealties or {}
4341 Babel.intrapealties['\csname bbl@sbcp@language\endcsname'] = #1
4342 Babel.locale_props[\the\localeid].intrapenalty = #1
4343 }}
4344 \begingroup
4345 \catcode`\%=12
4346 \catcode`\^=14
4347 \catcode`\'=12
4348 \catcode`\~=12
4349 \gdef\bbl@seaintraspace{^
4350 \let\bbl@seaintraspace\relax
4351 \directlua{
4352 Babel = Babel or {}
4353 Babel.sea_enabled = true
4354 Babel.sea_ranges = Babel.sea_ranges or {}
4355 function Babel.set_chrngs (script, chrng)
4356     local c = 0
4357     for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
4358         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
4359         c = c + 1
4360     end
4361 end
4362 function Babel.sea_disc_to_space (head)
4363     local sea_ranges = Babel.sea_ranges
4364     local last_char = nil
4365     local quad = 655360 ^^ 10 pt = 655360 = 10 * 65536
4366     for item in node.traverse(head) do
4367         local i = item.id
4368         if i == node.id'glyph' then
4369             last_char = item
4370         elseif i == 7 and item.subtype == 3 and last_char
4371             and last_char.char > 0x0C99 then
4372             quad = font.getfont(last_char.font).size
4373             for lg, rg in pairs(sea_ranges) do
4374                 if last_char.char > rg[1] and last_char.char < rg[2] then
4375                     lg = lg:sub(1, 4) ^^ Remove trailing number of, eg, Cyril1
4376                     local intraspace = Babel.intraspaces[lg]
4377                     local intrapenalty = Babel.intrapealties[lg]
4378                     local n
4379                     if intrapenalty ~= 0 then
4380                         n = node.new(14, 0) ^^ penalty
4381                         n.penalty = intrapenalty
4382                         node.insert_before(head, item, n)
4383                     end
4384                     n = node.new(12, 13) ^^ (glue, spaceskip)
4385                     node.setglue(n, intraspace.b * quad,
4386                         intraspace.p * quad,
4387                         intraspace.m * quad)
4388                     node.insert_before(head, item, n)
4389                     node.remove(head, item)
4390                 end
4391             end
4392         end
4393     end
4394 end
4395 }^^
4396 \bbl@luahyphenate}
4397 \catcode`\%=14

```

```

4398 \gdef\bbl@cjkintraspacespace{%
4399   \let\bbl@cjkintraspacespace\relax
4400   \directlua{
4401     Babel = Babel or {}
4402     require'babel-data-cjk.lua'
4403     Babel.cjk_enabled = true
4404     function Babel.cjk_linebreak(head)
4405       local GLYPH = node.id'glyph'
4406       local last_char = nil
4407       local quad = 655360      % 10 pt = 655360 = 10 * 65536
4408       local last_class = nil
4409       local last_lang = nil
4410
4411       for item in node.traverse(head) do
4412         if item.id == GLYPH then
4413
4414           local lang = item.lang
4415
4416           local LOCALE = node.get_attribute(item,
4417             luatexbase.registernumber'bbl@attr@locale')
4418           local props = Babel.locale_props[LOCALE]
4419
4420           local class = Babel.cjk_class[item.char].c
4421
4422           if class == 'cp' then class = 'cl' end % ]) as CL
4423           if class == 'id' then class = 'I' end
4424
4425           local br = 0
4426           if class and last_class and Babel.cjk_breaks[last_class][class] then
4427             br = Babel.cjk_breaks[last_class][class]
4428           end
4429
4430           if br == 1 and props.linebreak == 'c' and
4431             lang ~= \the\l@nohyphenation\space and
4432             last_lang ~= \the\l@nohyphenation then
4433             local intrapenalty = props.intrapenalty
4434             if intrapenalty ~= 0 then
4435               local n = node.new(14, 0)      % penalty
4436               n.penalty = intrapenalty
4437               node.insert_before(head, item, n)
4438             end
4439             local intraspacespace = props.intraspacespace
4440             local n = node.new(12, 13)      % (glue, spaceskip)
4441             node.setglue(n, intraspacespace.b * quad,
4442               intraspacespace.p * quad,
4443               intraspacespace.m * quad)
4444             node.insert_before(head, item, n)
4445           end
4446
4447           quad = font.getfont(item.font).size
4448           last_class = class
4449           last_lang = lang
4450         else % if penalty, glue or anything else
4451           last_class = nil
4452         end
4453       end
4454       lang.hyphenate(head)
4455     end
4456   }%

```

```

4457 \bbl@luahyphenate}
4458 \gdef\bbl@luahyphenate{%
4459 \let\bbl@luahyphenate\relax
4460 \directlua{
4461   luatexbase.add_to_callback('hyphenate',
4462     function (head, tail)
4463       if Babel.linebreaking.before then
4464         for k, func in ipairs(Babel.linebreaking.before) do
4465           func(head)
4466         end
4467       end
4468       if Babel.cjk_enabled then
4469         Babel.cjk_linebreak(head)
4470       end
4471       lang.hyphenate(head)
4472       if Babel.linebreaking.after then
4473         for k, func in ipairs(Babel.linebreaking.after) do
4474           func(head)
4475         end
4476       end
4477       if Babel.sea_enabled then
4478         Babel.sea_disc_to_space(head)
4479       end
4480     end,
4481     'Babel.hyphenate')
4482 }
4483 }
4484 \endgroup
4485 \def\bbl@provide@intraspace{%
4486 \bbl@ifunset{\bbl@intsp@language}{}%
4487 {\xexpandafter\ifx\csname bbl@intsp@language\endcsname\@empty\else
4488 \bbl@xin@{\bbl@cl{lnbrk}}{c}%
4489 \ifin@ % cjk
4490 \bbl@cjk@intraspace
4491 \directlua{
4492   Babel = Babel or {}
4493   Babel.locale_props = Babel.locale_props or {}
4494   Babel.locale_props[\the\localeid].linebreak = 'c'
4495 }%
4496 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}{\@}%
4497 \ifx\bbl@KVP@intrapenalty\@nil
4498 \bbl@intrapenalty0\@
4499 \fi
4500 \else % sea
4501 \bbl@sea@intraspace
4502 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}{\@}%
4503 \directlua{
4504   Babel = Babel or {}
4505   Babel.sea_ranges = Babel.sea_ranges or {}
4506   Babel.set_chranges('\bbl@cl{sbc}',
4507                     '\bbl@cl{chrng}')
4508 }%
4509 \ifx\bbl@KVP@intrapenalty\@nil
4510 \bbl@intrapenalty0\@
4511 \fi
4512 \fi
4513 \fi
4514 \ifx\bbl@KVP@intrapenalty\@nil\else
4515 \xexpandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@

```

4516      \fi}}

## 15.5 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.

```
4517 \AddBabelHook{luatex}{loadkernel}{%
4518 <<Restore Unicode catcodes before loading patterns>>}}
4519 \ifx\DisableBabelHook\undefined\endinput\fi
4520 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4521 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4522 \DisableBabelHook{babel-fontspec}
4523 <<Font selection>>
```

## 15.6 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 form a script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the `\language` and the `\localeid` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with / maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```
4524 \directlua{
4525 Babel.script_blocks = {
4526   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
4527              {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
4528   ['Armn'] = {{0x0530, 0x058F}},
4529   ['Beng'] = {{0x0980, 0x09FF}},
4530   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0ABBF}},
4531   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
4532   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
4533              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
4534   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
4535   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
4536              {0xAB00, 0xAB2F}},
4537   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
4538   % Don't follow strictly Unicode, which places some Coptic letters in
4539   % the 'Greek and Coptic' block
4540   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
4541   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
4542              {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
4543              {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
4544              {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
4545              {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
4546              {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
4547   ['Hebr'] = {{0x0590, 0x05FF}},
4548   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF}},
```

```

4549         {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
4550 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
4551 ['Knda'] = {{0x0C80, 0x0CFF}},
4552 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
4553             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
4554             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
4555 ['Lao'] = {{0x0E80, 0x0EFF}},
4556 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
4557             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
4558             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
4559 ['Mahj'] = {{0x11150, 0x1117F}},
4560 ['Mlym'] = {{0x0D00, 0x0D7F}},
4561 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
4562 ['Orya'] = {{0x0B00, 0x0B7F}},
4563 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
4564 ['Taml'] = {{0x0B80, 0x0BFF}},
4565 ['Telu'] = {{0x0C00, 0x0C7F}},
4566 ['Tfng'] = {{0x2D30, 0x2D7F}},
4567 ['Thai'] = {{0x0E00, 0x0E7F}},
4568 ['Tibt'] = {{0x0F00, 0x0FFF}},
4569 ['Vaii'] = {{0xA500, 0xA63F}},
4570 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
4571 }
4572
4573 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
4574 Babel.script_blocks.Hant = Babel.script_blocks.Hans
4575 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
4576
4577 function Babel.locale_map(head)
4578   if not Babel.locale_mapped then return head end
4579
4580   local LOCALE = luatexbase.registernumber'bb1@attr@locale'
4581   local GLYPH = node.id('glyph')
4582   local inmath = false
4583   local toloc_save
4584   for item in node.traverse(head) do
4585     local toloc
4586     if not inmath and item.id == GLYPH then
4587       % Optimization: build a table with the chars found
4588       if Babel.chr_to_loc[item.char] then
4589         toloc = Babel.chr_to_loc[item.char]
4590       else
4591         for lc, maps in pairs(Babel.loc_to_scr) do
4592           for _, rg in pairs(maps) do
4593             if item.char >= rg[1] and item.char <= rg[2] then
4594               Babel.chr_to_loc[item.char] = lc
4595               toloc = lc
4596               break
4597             end
4598           end
4599         end
4600       end
4601       % Now, take action, but treat composite chars in a different
4602       % fashion, because they 'inherit' the previous locale. Not yet
4603       % optimized.
4604       if not toloc and
4605         (item.char >= 0x0300 and item.char <= 0x036F) or
4606         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
4607         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then

```

```

4608         toloc = toloc_save
4609     end
4610     if toloc and toloc > -1 then
4611         if Babel.locale_props[toloc].lg then
4612             item.lang = Babel.locale_props[toloc].lg
4613             node.set_attribute(item, LOCALE, toloc)
4614         end
4615         if Babel.locale_props[toloc]['/'..item.font] then
4616             item.font = Babel.locale_props[toloc]['/'..item.font]
4617         end
4618         toloc_save = toloc
4619     end
4620     elseif not inmath and item.id == 7 then
4621         item.replace = item.replace and Babel.locale_map(item.replace)
4622         item.pre      = item.pre and Babel.locale_map(item.pre)
4623         item.post     = item.post and Babel.locale_map(item.post)
4624     elseif item.id == node.id'math' then
4625         inmath = (item.subtype == 0)
4626     end
4627 end
4628 return head
4629 end
4630 }

```

The code for `\babelcharproperty` is straightforward. Just note the modified lua table can be different.

```

4631 \newcommand\babelcharproperty[1]{%
4632   \count@=#1\relax
4633   \ifvmode
4634     \expandafter\bbl@chprop
4635   \else
4636     \bbl@error{\string\babelcharproperty\space can be used only in\\%
4637               vertical mode (preamble or between paragraphs)}%
4638     {See the manual for futher info}%
4639   \fi}
4640 \newcommand\bbl@chprop[3][\the\count@]{%
4641   \@tempcnta=#1\relax
4642   \bbl@ifunset{\bbl@chprop@#2}%
4643   {\bbl@error{No property named '#2'. Allowed values are\\%
4644             direction (bc), mirror (bmg), and linebreak (lb)}%
4645    {See the manual for futher info}}%
4646   }%
4647   \loop
4648     \bbl@cs{chprop@#2}{#3}%
4649   \ifnum\count@<\@tempcnta
4650     \advance\count@\@ne
4651   \repeat}
4652 \def\bbl@chprop@direction#1{%
4653   \directlua{
4654     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
4655     Babel.characters[\the\count@]['d'] = '#1'
4656   }}
4657 \let\bbl@chprop@bc\bbl@chprop@direction
4658 \def\bbl@chprop@mirror#1{%
4659   \directlua{
4660     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
4661     Babel.characters[\the\count@]['m'] = '\number#1'
4662   }}
4663 \let\bbl@chprop@bmg\bbl@chprop@mirror

```



```

4664 \def\bbl@chprop@linebreak#1{%
4665   \directlua{
4666     Babel.Babel.cjk_characters[\the\count@] = Babel.Babel.cjk_characters[\the\count@] or {}
4667     Babel.Babel.cjk_characters[\the\count@]['c'] = '#1'
4668   }}
4669 \let\bbl@chprop@lb\bbl@chprop@linebreak
4670 \def\bbl@chprop@locale#1{%
4671   \directlua{
4672     Babel.chr_to_loc = Babel.chr_to_loc or {}
4673     Babel.chr_to_loc[\the\count@] =
4674       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
4675   }}

```

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 `tex.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.

```

4676 \begingroup
4677 \catcode`\#=12
4678 \catcode`\%=12
4679 \catcode`\&=14
4680 \directlua{
4681   Babel.linebreaking.replacements = {}
4682
4683   function Babel.str_to_nodes(fn, matches, base)
4684     local n, head, last
4685     if fn == nil then return nil end
4686     for s in string.utfvalues(fn(matches)) do
4687       if base.id == 7 then
4688         base = base.replace
4689       end
4690       n = node.copy(base)
4691       n.char = s
4692       if not head then
4693         head = n
4694       else
4695         last.next = n
4696       end
4697       last = n
4698     end
4699     return head
4700   end
4701
4702   function Babel.fetch_word(head, funct)
4703     local word_string = ''
4704     local word_nodes = {}
4705     local lang
4706     local item = head
4707

```

```

4708     while item do
4709
4710         if item.id == 29
4711             and not(item.char == 124) && ie, not |
4712             and not(item.char == 61) && ie, not =
4713             and (item.lang == lang or lang == nil) then
4714                 lang = lang or item.lang
4715                 word_string = word_string .. unicode.utf8.char(item.char)
4716                 word_nodes[#word_nodes+1] = item
4717
4718             elseif item.id == 7 and item.subtype == 2 then
4719                 word_string = word_string .. '='
4720                 word_nodes[#word_nodes+1] = item
4721
4722             elseif item.id == 7 and item.subtype == 3 then
4723                 word_string = word_string .. '|'
4724                 word_nodes[#word_nodes+1] = item
4725
4726             elseif word_string == '' then
4727                 && pass
4728
4729             else
4730                 return word_string, word_nodes, item, lang
4731             end
4732
4733             item = item.next
4734         end
4735     end
4736
4737     function Babel.post_hyphenate_replace(head)
4738         local u = unicode.utf8
4739         local lbkr = Babel.linebreaking.replacements
4740         local word_head = head
4741
4742         while true do
4743             local w, wn, nw, lang = Babel.fetch_word(word_head)
4744             if not lang then return head end
4745
4746             if not lbkr[lang] then
4747                 break
4748             end
4749
4750             for k=1, #lbkr[lang] do
4751                 local p = lbkr[lang][k].pattern
4752                 local r = lbkr[lang][k].replace
4753
4754                 while true do
4755                     local matches = { u.match(w, p) }
4756                     if #matches < 2 then break end
4757
4758                     local first = table.remove(matches, 1)
4759                     local last = table.remove(matches, #matches)
4760
4761                     && Fix offsets, from bytes to unicode.
4762                     first = u.len(w:sub(1, first-1)) + 1
4763                     last = u.len(w:sub(1, last-1))
4764
4765                     local new && used when inserting and removing nodes
4766                     local changed = 0

```

```

4767
4768      %% This loop traverses the replace list and takes the
4769      %% corresponding actions
4770      for q = first, last do
4771          local crep = r[q-first+1]
4772          local char_node = wn[q]
4773          local char_base = char_node
4774
4775          if crep and crep.data then
4776              char_base = wn[crep.data+first-1]
4777          end
4778
4779          if crep == {} then
4780              break
4781          elseif crep == nil then
4782              changed = changed + 1
4783              node.remove(head, char_node)
4784          elseif crep and (crep.pre or crep.no or crep.post) then
4785              changed = changed + 1
4786              d = node.new(7, 0)    %% (disc, discretionary)
4787              d.pre = Babel.str_to_nodes(crep.pre, matches, char_base)
4788              d.post = Babel.str_to_nodes(crep.post, matches, char_base)
4789              d.replace = Babel.str_to_nodes(crep.no, matches, char_base)
4790              d.attr = char_base.attr
4791              if crep.pre == nil then    %% TeXbook p96
4792                  d.penalty = crep.penalty or tex.hyphenpenalty
4793              else
4794                  d.penalty = crep.penalty or tex.exhyphenpenalty
4795              end
4796              head, new = node.insert_before(head, char_node, d)
4797              node.remove(head, char_node)
4798              if q == 1 then
4799                  word_head = new
4800              end
4801          elseif crep and crep.string then
4802              changed = changed + 1
4803              local str = crep.string(matches)
4804              if str == '' then
4805                  if q == 1 then
4806                      word_head = char_node.next
4807                  end
4808                  head, new = node.remove(head, char_node)
4809              elseif char_node.id == 29 and u.len(str) == 1 then
4810                  char_node.char = string.utfvalue(str)
4811              else
4812                  local n
4813                  for s in string.utfvalues(str) do
4814                      if char_node.id == 7 then
4815                          log('Automatic hyphens cannot be replaced, just removed.')
4816                      else
4817                          n = node.copy(char_base)
4818                      end
4819                      n.char = s
4820                      if q == 1 then
4821                          head, new = node.insert_before(head, char_node, n)
4822                          word_head = new
4823                      else
4824                          node.insert_before(head, char_node, n)
4825                      end
4826                  end
4827              end
4828          end
4829      end

```

```

4826             end
4827
4828             node.remove(head, char_node)
4829         end %% string length
4830     end %% if char and char.string
4831 end %% for char in match
4832 if changed > 20 then
4833     texio.write('Too many changes. Ignoring the rest.')
4834 elseif changed > 0 then
4835     w, wn, nw = Babel.fetch_word(word_head)
4836 end
4837
4838     end %% for match
4839 end %% for patterns
4840 word_head = nw
4841 end %% for words
4842 return head
4843 end
4844
4845 %% The following functions belong to the next macro
4846
4847 %% This table stores capture maps, numbered consecutively
4848 Babel.capture_maps = {}
4849
4850 function Babel.capture_func(key, cap)
4851     local ret = "[" .. cap:gsub('{{([0-9])}}', "]]..m[%1]..[(") .. "]"
4852     ret = ret:gsub('{{([0-9])|([^\]|+)|(-)}}', Babel.capture_func_map)
4853     ret = ret:gsub("%[%[%]%.%", '')
4854     ret = ret:gsub("%.%.%.%.%", '')
4855     return key .. "[=function(m) return ]] .. ret .. [[ end]]
4856 end
4857
4858 function Babel.capt_map(from, mapno)
4859     return Babel.capture_maps[mapno][from] or from
4860 end
4861
4862 %% Handle the {n|abc|ABC} syntax in captures
4863 function Babel.capture_func_map(capno, from, to)
4864     local froms = {}
4865     for s in string.utfcharacters(from) do
4866         table.insert(froms, s)
4867     end
4868     local cnt = 1
4869     table.insert(Babel.capture_maps, {})
4870     local mlen = table.getn(Babel.capture_maps)
4871     for s in string.utfcharacters(to) do
4872         Babel.capture_maps[mlen][froms[cnt]] = s
4873         cnt = cnt + 1
4874     end
4875     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
4876         (mlen) .. ").." .. "["
4877 end
4878
4879 }

```

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,  $\text{pre}=\{1\}\{1\}$ - becomes `function(m) return m[1]..m[1]..'-' end`, where  $m$  are the matches returned after applying the pattern. With a mapped capture the functions

are similar to `function(m) return Babel.capt_map(m[1],1) end`, where the last argument identifies the mapping to be applied to `m[1]`. The way it is carried out is somewhat tricky, but the effect is not dissimilar to `lua load` – save the code as string in a TeX macro, and expand this macro at the appropriate place. As `\directlua` does not take into account the current catcode of `@`, we just avoid this character in macro names (which explains the internal group, too).

```

4880 \catcode`\#=6
4881 \gdef\babelposthyphenation#1#2#3{&%
4882   \bbl@activateposthyphen
4883   \begingroup
4884     \def\babeltempa{\bbl@add@list\babeltempb}&%
4885     \let\babeltempb\@empty
4886     \bbl@foreach{#3}{&%
4887       \bbl@ifsamestring{##1}{remove}&%
4888       {\bbl@add@list\babeltempb{nil}}&%
4889       {\directlua{
4890         local rep = [[##1]]
4891         rep = rep:gsub(  '(no)%s*=%s*([^\s,]*)', Babel.capture_func)
4892         rep = rep:gsub(  '(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
4893         rep = rep:gsub(  '(post)%s*=%s*([^\s,]*)', Babel.capture_func)
4894         rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
4895         tex.print([[string\babeltempa{}}] .. rep .. [[}}]])
4896       }}&%
4897     \directlua{
4898       local lbkr = Babel.linebreaking.replacements
4899       local u = unicode.utf8
4900       &% Convert pattern:
4901       local patt = string.gsub([[#2]], '%s', '')
4902       if not u.find(patt, '()', nil, true) then
4903         patt = '()' .. patt .. '()'
4904       end
4905       patt = u.gsub(patt, '{(.)}',
4906         function (n)
4907           return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
4908         end)
4909       lbkr[\the\csname l@#1\endcsname] = lbkr[\the\csname l@#1\endcsname] or {}
4910       table.insert(lbkr[\the\csname l@#1\endcsname],
4911         { pattern = patt, replace = { \babeltempb } })
4912     }&%
4913   \endgroup}
4914 \endgroup
4915 \def\bbl@activateposthyphen{%
4916   \let\bbl@activateposthyphen\relax
4917   \directlua{
4918     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
4919   }}

```

## 15.7 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.

```

4920 \bbl@trace{Redefinitions for bidi layout}
4921 \ifx\@eqnnum\undefined\else
4922   \ifx\bbl@attr@dir\undefined\else
4923     \edef\@eqnnum{%
4924       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
4925       \unexpanded\expandafter{\@eqnnum}}%
4926   \fi
4927 \fi
4928 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
4929 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4930   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
4931     \bbl@exp{%
4932       \mathdir\the\bodydir
4933       #1%           Once entered in math, set boxes to restore values
4934       \<ifmmode>%
4935       \everyvbox{%
4936         \the\everyvbox
4937         \bodydir\the\bodydir
4938         \mathdir\the\mathdir
4939         \everyhbox{\the\everyhbox}%
4940         \everyvbox{\the\everyvbox}}%
4941       \everyhbox{%
4942         \the\everyhbox
4943         \bodydir\the\bodydir
4944         \mathdir\the\mathdir
4945         \everyhbox{\the\everyhbox}%
4946         \everyvbox{\the\everyvbox}}%
4947       \<fi>}}%
4948   \def\@hangfrom#1{%
4949     \setbox\@tempboxa\hbox{{#1}}%
4950     \hangindent\wd\@tempboxa
4951     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
4952       \shapemode\@ne
4953     \fi
4954     \noindent\box\@tempboxa}
4955 \fi
4956 \IfBabelLayout{tabular}
4957   {\let\bbl@OL@tabular\@tabular
4958     \bbl@replace\@tabular{$}{\bbl@nextfake$}%
4959     \let\bbl@NL@tabular\@tabular
4960     \AtBeginDocument{%
4961       \ifx\bbl@NL@tabular\@tabular\else
4962         \bbl@replace\@tabular{$}{\bbl@nextfake$}%
4963         \let\bbl@NL@tabular\@tabular
4964       \fi}}
4965   {}
4966 \IfBabelLayout{lists}
4967   {\let\bbl@OL@list\list
4968     \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
4969     \let\bbl@NL@list\list
4970     \def\bbl@listparshape#1#2#3{%
4971       \parshape #1 #2 #3 %
4972       \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else

```

```

4973     \shapemode\tw@
4974     \fi}}
4975 {}
4976 \IfBabelLayout{graphics}
4977 {\let\bbbl@pictresetdir\relax
4978  \def\bbbl@pictsetdir{%
4979   \ifcase\bbbl@thetextdir
4980   \let\bbbl@pictresetdir\relax
4981   \else
4982   \textdir TLT\relax
4983   \def\bbbl@pictresetdir{\textdir TRT\relax}%
4984   \fi}%
4985 \let\bbbl@OL@@picture\@picture
4986 \let\bbbl@OL@put\put
4987 \bbbl@sreplace\@picture{\hskip-}\bbbl@pictsetdir\hskip-}%
4988 \def\put(#1,#2)#3{% Not easy to patch. Better redefine.
4989  \@killglue
4990  \raise#2\unitlength
4991  \hb@xt@z@{\kern#1\unitlength\bbbl@pictresetdir#3\hss}}%
4992 \AtBeginDocument
4993 {\ifx\tikz@atbegin@node\@undefined\else
4994  \let\bbbl@OL@pgfpicture\pgfpicture
4995  \bbbl@sreplace\pgfpicture{\pgfpicturetrue}\bbbl@pictsetdir\pgfpicturetrue}%
4996  \bbbl@add\pgfsys@beginpicture{\bbbl@pictsetdir}%
4997  \bbbl@add\tikz@atbegin@node{\bbbl@pictresetdir}%
4998  \fi}}
4999 {}

```

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.

```

5000 \IfBabelLayout{counters}%
5001 {\let\bbbl@OL@@textsuperscript\@textsuperscript
5002  \bbbl@sreplace\@textsuperscript{\m@th}\m@th\mathdir\pagedir}%
5003 \let\bbbl@latinarabic=\@arabic
5004 \let\bbbl@OL@@arabic\@arabic
5005 \def\@arabic#1{\babelsublr{\bbbl@latinarabic#1}}%
5006 \@ifpackagewith{babel}{bidi=default}%
5007 {\let\bbbl@asciroman=\@roman
5008  \let\bbbl@OL@@roman\@roman
5009  \def\@roman#1{\babelsublr{\ensureascii\bbbl@asciroman#1}}}%
5010 \let\bbbl@asciiRoman=\@Roman
5011 \let\bbbl@OL@@roman\@Roman
5012 \def\@Roman#1{\babelsublr{\ensureascii\bbbl@asciiRoman#1}}}%
5013 \let\bbbl@OL@labelenumii\labelenumii
5014 \def\labelenumii{}\theenumii{}%
5015 \let\bbbl@OL@p@enumiii\p@enumiii
5016 \def\p@enumiii{\p@enumii}\theenumii{}\}\}\}\}
5017 <<Footnote changes>>
5018 \IfBabelLayout{footnotes}%
5019 {\let\bbbl@OL@footnote\footnote
5020  \BabelFootnote\footnote\languagename{}\}\}%
5021  \BabelFootnote\localfootnote\languagename{}\}\}%
5022  \BabelFootnote\mainfootnote{}\}\}\}\}
5023 {}

```

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.

```

5024 \IfBabelLayout{extras}%

```

```

5025 {\let\bbl@OL@underline\underline
5026 \bbl@sreplace\underline{\$@@underline}{\bbl@nextfake\$@@underline}%
5027 \let\bbl@OL@LaTeX2e\LaTeX2e
5028 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
5029 \if b\expandafter\car\@series\@nil\boldmath\fi
5030 \babelsublr{%
5031 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
5032 {}
5033 \luatex

```

## 15.8 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.

```

5034 (*basic-r)
5035 Babel = Babel or {}
5036

```



```

5037 Babel.bidi_enabled = true
5038
5039 require('babel-data-bidi.lua')
5040
5041 local characters = Babel.characters
5042 local ranges = Babel.ranges
5043
5044 local DIR = node.id("dir")
5045
5046 local function dir_mark(head, from, to, outer)
5047   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
5048   local d = node.new(DIR)
5049   d.dir = '+' .. dir
5050   node.insert_before(head, from, d)
5051   d = node.new(DIR)
5052   d.dir = '-' .. dir
5053   node.insert_after(head, to, d)
5054 end
5055
5056 function Babel.bidi(head, ispar)
5057   local first_n, last_n      -- first and last char with nums
5058   local last_es              -- an auxiliary 'last' used with nums
5059   local first_d, last_d      -- first and last char in L/R block
5060   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):

```

5061   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
5062   local strong_lr = (strong == 'l') and 'l' or 'r'
5063   local outer = strong
5064
5065   local new_dir = false
5066   local first_dir = false
5067   local inmath = false
5068
5069   local last_lr
5070
5071   local type_n = ''
5072
5073   for item in node.traverse(head) do
5074
5075     -- three cases: glyph, dir, otherwise
5076     if item.id == node.id'glyph'
5077       or (item.id == 7 and item.subtype == 2) then
5078
5079       local itemchar
5080       if item.id == 7 and item.subtype == 2 then
5081         itemchar = item.replace.char
5082       else
5083         itemchar = item.char
5084       end
5085       local chardata = characters[itemchar]
5086       dir = chardata and chardata.d or nil
5087       if not dir then
5088         for nn, et in ipairs(ranges) do
5089           if itemchar < et[1] then
5090             break
5091           elseif itemchar <= et[2] then

```

```

5092         dir = et[3]
5093         break
5094     end
5095 end
5096 end
5097 dir = dir or 'l'
5098 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).

```

5099     if new_dir then
5100         attr_dir = 0
5101         for at in node.traverse(item.attr) do
5102             if at.number == luatexbase.registernumber'bbl@attr@dir' then
5103                 attr_dir = at.value % 3
5104             end
5105         end
5106         if attr_dir == 1 then
5107             strong = 'r'
5108         elseif attr_dir == 2 then
5109             strong = 'al'
5110         else
5111             strong = 'l'
5112         end
5113         strong_lr = (strong == 'l') and 'l' or 'r'
5114         outer = strong_lr
5115         new_dir = false
5116     end
5117
5118     if dir == 'nsm' then dir = strong end -- W1

```

**Numbers.** The dual <al>/<r> system for R is somewhat cumbersome.

```

5119     dir_real = dir -- We need dir_real to set strong below
5120     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:

```

5121     if strong == 'al' then
5122         if dir == 'en' then dir = 'an' end -- W2
5123         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
5124         strong_lr = 'r' -- W3
5125     end

```

Once finished the basic setup for glyphs, consider the two other cases: dir node and the rest.

```

5126     elseif item.id == node.id'dir' and not inmath then
5127         new_dir = true
5128         dir = nil
5129     elseif item.id == node.id'math' then
5130         inmath = (item.subtype == 0)
5131     else
5132         dir = nil -- Not a char
5133     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>.

```

5134   if dir == 'en' or dir == 'an' or dir == 'et' then
5135       if dir ~= 'et' then
5136           type_n = dir
5137       end
5138       first_n = first_n or item
5139       last_n = last_es or item
5140       last_es = nil
5141   elseif dir == 'es' and last_n then -- W3+W6
5142       last_es = item
5143   elseif dir == 'cs' then             -- it's right - do nothing
5144   elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
5145       if strong_lr == 'r' and type_n ~= '' then
5146           dir_mark(head, first_n, last_n, 'r')
5147       elseif strong_lr == 'l' and first_d and type_n == 'an' then
5148           dir_mark(head, first_n, last_n, 'r')
5149           dir_mark(head, first_d, last_d, outer)
5150           first_d, last_d = nil, nil
5151       elseif strong_lr == 'l' and type_n ~= '' then
5152           last_d = last_n
5153       end
5154       type_n = ''
5155       first_n, last_n = nil, nil
5156   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:

```

5157   if dir == 'l' or dir == 'r' then
5158       if dir ~= outer then
5159           first_d = first_d or item
5160           last_d = item
5161       elseif first_d and dir ~= strong_lr then
5162           dir_mark(head, first_d, last_d, outer)
5163           first_d, last_d = nil, nil
5164       end
5165   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.

```

5166   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
5167       item.char = characters[item.char] and
5168           characters[item.char].m or item.char
5169   elseif (dir or new_dir) and last_lr ~= item then
5170       local mir = outer .. strong_lr .. (dir or outer)
5171       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
5172           for ch in node.traverse(node.next(last_lr)) do
5173               if ch == item then break end
5174               if ch.id == node.id'glyph' and characters[ch.char] then
5175                   ch.char = characters[ch.char].m or ch.char
5176               end

```

```

5177         end
5178     end
5179 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).

```

5180     if dir == 'l' or dir == 'r' then
5181         last_lr = item
5182         strong = dir_real          -- Don't search back - best save now
5183         strong_lr = (strong == 'l') and 'l' or 'r'
5184     elseif new_dir then
5185         last_lr = nil
5186     end
5187 end

```

Mirror the last chars if they are no directed. And make sure any open block is closed, too.

```

5188     if last_lr and outer == 'r' then
5189         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
5190             if characters[ch.char] then
5191                 ch.char = characters[ch.char].m or ch.char
5192             end
5193         end
5194     end
5195     if first_n then
5196         dir_mark(head, first_n, last_n, outer)
5197     end
5198     if first_d then
5199         dir_mark(head, first_d, last_d, outer)
5200     end

```

In boxes, the dir node could be added before the original head, so the actual head is the previous node.

```

5201     return node.prev(head) or head
5202 end
5203 </basic-r>

```

And here the Lua code for bidi=basic:

```

5204 <(*basic>
5205 Babel = Babel or {}
5206
5207 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
5208
5209 Babel.fontmap = Babel.fontmap or {}
5210 Babel.fontmap[0] = {}          -- l
5211 Babel.fontmap[1] = {}          -- r
5212 Babel.fontmap[2] = {}          -- al/an
5213
5214 Babel.bidi_enabled = true
5215 Babel.mirroring_enabled = true
5216
5217 require('babel-data-bidi.lua')
5218
5219 local characters = Babel.characters
5220 local ranges = Babel.ranges
5221
5222 local DIR = node.id('dir')
5223 local GLYPH = node.id('glyph')
5224
5225 local function insert_implicit(head, state, outer)

```

```

5226 local new_state = state
5227 if state.sim and state.eim and state.sim ~= state.eim then
5228     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
5229     local d = node.new(DIR)
5230     d.dir = '+' .. dir
5231     node.insert_before(head, state.sim, d)
5232     local d = node.new(DIR)
5233     d.dir = '-' .. dir
5234     node.insert_after(head, state.eim, d)
5235 end
5236 new_state.sim, new_state.eim = nil, nil
5237 return head, new_state
5238 end
5239
5240 local function insert_numeric(head, state)
5241     local new
5242     local new_state = state
5243     if state.san and state.ean and state.san ~= state.ean then
5244         local d = node.new(DIR)
5245         d.dir = '+TLT'
5246         _, new = node.insert_before(head, state.san, d)
5247         if state.san == state.sim then state.sim = new end
5248         local d = node.new(DIR)
5249         d.dir = '-TLT'
5250         _, new = node.insert_after(head, state.ean, d)
5251         if state.ean == state.eim then state.eim = new end
5252     end
5253     new_state.san, new_state.ean = nil, nil
5254     return head, new_state
5255 end
5256
5257 -- TODO - \hbox with an explicit dir can lead to wrong results
5258 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
5259 -- was s made to improve the situation, but the problem is the 3-dir
5260 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
5261 -- well.
5262
5263 function Babel.bidi(head, ispar, hdir)
5264     local d -- d is used mainly for computations in a loop
5265     local prev_d = ''
5266     local new_d = false
5267
5268     local nodes = {}
5269     local outer_first = nil
5270     local inmath = false
5271
5272     local glue_d = nil
5273     local glue_i = nil
5274
5275     local has_en = false
5276     local first_et = nil
5277
5278     local ATDIR = luatexbase.registernumber'bbl@attr@dir'
5279
5280     local save_outer
5281     local temp = node.get_attribute(head, ATDIR)
5282     if temp then
5283         temp = temp % 3
5284         save_outer = (temp == 0 and 'l') or

```

```

5285             (temp == 1 and 'r') or
5286             (temp == 2 and 'al')
5287 elseif ispar then -- Or error? Shouldn't happen
5288     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
5289 else -- Or error? Shouldn't happen
5290     save_outer = ('TRT' == hdir) and 'r' or 'l'
5291 end
5292 -- when the callback is called, we are just _after_ the box,
5293 -- and the textdir is that of the surrounding text
5294 -- if not ispar and hdir ~= tex.textdir then
5295 --     save_outer = ('TRT' == hdir) and 'r' or 'l'
5296 -- end
5297 local outer = save_outer
5298 local last = outer
5299 -- 'al' is only taken into account in the first, current loop
5300 if save_outer == 'al' then save_outer = 'r' end
5301
5302 local fontmap = Babel.fontmap
5303
5304 for item in node.traverse(head) do
5305
5306     -- In what follows, #node is the last (previous) node, because the
5307     -- current one is not added until we start processing the neutrals.
5308
5309     -- three cases: glyph, dir, otherwise
5310     if item.id == GLYPH
5311         or (item.id == 7 and item.subtype == 2) then
5312
5313         local d_font = nil
5314         local item_r
5315         if item.id == 7 and item.subtype == 2 then
5316             item_r = item.replace -- automatic discs have just 1 glyph
5317         else
5318             item_r = item
5319         end
5320         local chardata = characters[item_r.char]
5321         d = chardata and chardata.d or nil
5322         if not d or d == 'nsm' then
5323             for nn, et in ipairs(ranges) do
5324                 if item_r.char < et[1] then
5325                     break
5326                 elseif item_r.char <= et[2] then
5327                     if not d then d = et[3]
5328                     elseif d == 'nsm' then d_font = et[3]
5329                     end
5330                     break
5331                 end
5332             end
5333         end
5334         d = d or 'l'
5335
5336         -- A short 'pause' in bidi for mapfont
5337         d_font = d_font or d
5338         d_font = (d_font == 'l' and 0) or
5339             (d_font == 'nsm' and 0) or
5340             (d_font == 'r' and 1) or
5341             (d_font == 'al' and 2) or
5342             (d_font == 'an' and 2) or nil
5343         if d_font and fontmap and fontmap[d_font][item_r.font] then

```

```

5344     item_r.font = fontmap[d_font][item_r.font]
5345 end
5346
5347 if new_d then
5348     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
5349     if inmath then
5350         attr_d = 0
5351     else
5352         attr_d = node.get_attribute(item, ATDIR)
5353         attr_d = attr_d % 3
5354     end
5355     if attr_d == 1 then
5356         outer_first = 'r'
5357         last = 'r'
5358     elseif attr_d == 2 then
5359         outer_first = 'r'
5360         last = 'al'
5361     else
5362         outer_first = 'l'
5363         last = 'l'
5364     end
5365     outer = last
5366     has_en = false
5367     first_et = nil
5368     new_d = false
5369 end
5370
5371 if glue_d then
5372     if (d == 'l' and 'l' or 'r') ~= glue_d then
5373         table.insert(nodes, {glue_i, 'on', nil})
5374     end
5375     glue_d = nil
5376     glue_i = nil
5377 end
5378
5379 elseif item.id == DIR then
5380     d = nil
5381     new_d = true
5382
5383 elseif item.id == node.id'glue' and item.subtype == 13 then
5384     glue_d = d
5385     glue_i = item
5386     d = nil
5387
5388 elseif item.id == node.id'math' then
5389     inmath = (item.subtype == 0)
5390
5391 else
5392     d = nil
5393 end
5394
5395 -- AL <= EN/ET/ES      -- W2 + W3 + W6
5396 if last == 'al' and d == 'en' then
5397     d = 'an'           -- W3
5398 elseif last == 'al' and (d == 'et' or d == 'es') then
5399     d = 'on'           -- W6
5400 end
5401
5402 -- EN + CS/ES + EN      -- W4

```

```

5403 if d == 'en' and #nodes >= 2 then
5404     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
5405         and nodes[#nodes-1][2] == 'en' then
5406         nodes[#nodes][2] = 'en'
5407     end
5408 end
5409
5410 -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
5411 if d == 'an' and #nodes >= 2 then
5412     if (nodes[#nodes][2] == 'cs')
5413         and nodes[#nodes-1][2] == 'an' then
5414         nodes[#nodes][2] = 'an'
5415     end
5416 end
5417
5418 -- ET/EN                -- W5 + W7->l / W6->on
5419 if d == 'et' then
5420     first_et = first_et or (#nodes + 1)
5421 elseif d == 'en' then
5422     has_en = true
5423     first_et = first_et or (#nodes + 1)
5424 elseif first_et then    -- d may be nil here !
5425     if has_en then
5426         if last == 'l' then
5427             temp = 'l'    -- W7
5428         else
5429             temp = 'en'   -- W5
5430         end
5431     else
5432         temp = 'on'      -- W6
5433     end
5434     for e = first_et, #nodes do
5435         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
5436     end
5437     first_et = nil
5438     has_en = false
5439 end
5440
5441 if d then
5442     if d == 'al' then
5443         d = 'r'
5444         last = 'al'
5445     elseif d == 'l' or d == 'r' then
5446         last = d
5447     end
5448     prev_d = d
5449     table.insert(nodes, {item, d, outer_first})
5450 end
5451
5452 outer_first = nil
5453
5454 end
5455
5456 -- TODO -- repeated here in case EN/ET is the last node. Find a
5457 -- better way of doing things:
5458 if first_et then    -- dir may be nil here !
5459     if has_en then
5460         if last == 'l' then
5461             temp = 'l'    -- W7

```



```

5462     else
5463         temp = 'en'    -- W5
5464     end
5465     else
5466         temp = 'on'    -- W6
5467     end
5468     for e = first_et, #nodes do
5469         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
5470     end
5471 end
5472
5473 -- dummy node, to close things
5474 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
5475
5476 ----- NEUTRAL -----
5477
5478 outer = save_outer
5479 last = outer
5480
5481 local first_on = nil
5482
5483 for q = 1, #nodes do
5484     local item
5485
5486     local outer_first = nodes[q][3]
5487     outer = outer_first or outer
5488     last = outer_first or last
5489
5490     local d = nodes[q][2]
5491     if d == 'an' or d == 'en' then d = 'r' end
5492     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
5493
5494     if d == 'on' then
5495         first_on = first_on or q
5496     elseif first_on then
5497         if last == d then
5498             temp = d
5499         else
5500             temp = outer
5501         end
5502         for r = first_on, q - 1 do
5503             nodes[r][2] = temp
5504             item = nodes[r][1]    -- MIRRORING
5505             if Babel.mirroring_enabled and item.id == GLYPH
5506                 and temp == 'r' and characters[item.char] then
5507                 local font_mode = font.fonts[item.font].properties.mode
5508                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
5509                     item.char = characters[item.char].m or item.char
5510                 end
5511             end
5512         end
5513         first_on = nil
5514     end
5515
5516     if d == 'r' or d == 'l' then last = d end
5517 end
5518
5519 ----- IMPLICIT, REORDER -----
5520

```

```

5521 outer = save_outer
5522 last = outer
5523
5524 local state = {}
5525 state.has_r = false
5526
5527 for q = 1, #nodes do
5528
5529     local item = nodes[q][1]
5530
5531     outer = nodes[q][3] or outer
5532
5533     local d = nodes[q][2]
5534
5535     if d == 'nsm' then d = last end          -- W1
5536     if d == 'en' then d = 'an' end
5537     local isdir = (d == 'r' or d == 'l')
5538
5539     if outer == 'l' and d == 'an' then
5540         state.san = state.san or item
5541         state.ean = item
5542     elseif state.san then
5543         head, state = insert_numeric(head, state)
5544     end
5545
5546     if outer == 'l' then
5547         if d == 'an' or d == 'r' then      -- im -> implicit
5548             if d == 'r' then state.has_r = true end
5549             state.sim = state.sim or item
5550             state.eim = item
5551         elseif d == 'l' and state.sim and state.has_r then
5552             head, state = insert_implicit(head, state, outer)
5553         elseif d == 'l' then
5554             state.sim, state.eim, state.has_r = nil, nil, false
5555         end
5556     else
5557         if d == 'an' or d == 'l' then
5558             if nodes[q][3] then -- nil except after an explicit dir
5559                 state.sim = item -- so we move sim 'inside' the group
5560             else
5561                 state.sim = state.sim or item
5562             end
5563             state.eim = item
5564         elseif d == 'r' and state.sim then
5565             head, state = insert_implicit(head, state, outer)
5566         elseif d == 'r' then
5567             state.sim, state.eim = nil, nil
5568         end
5569     end
5570
5571     if isdir then
5572         last = d          -- Don't search back - best save now
5573     elseif d == 'on' and state.san then
5574         state.san = state.san or item
5575         state.ean = item
5576     end
5577
5578 end
5579

```

```

5580 return node.prev(head) or head
5581 end
5582 </basic>

```

## 16 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.

## 17 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.

```

5583 <{*nil}
5584 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language]
5585 \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.

```

5586 \ifx\l@nil\undefined
5587 \newlanguage\l@nil
5588 \@namedef{bbl@hyphendata@the\l@nil}{}{}{}% Remove warning
5589 \let\bbl@elt\relax
5590 \edef\bbl@languages{% Add it to the list of languages
5591 \bbl@languages\bbl@elt{nil}{the\l@nil}{}{}}
5592 \fi

```

This macro is used to store the values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```

5593 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

The next step consists of defining commands to switch to (and from) the ‘nil’ language.

```

\captionnil
\datenil
5594 \let\captionnil\@empty
5595 \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.

```

5596 \ldf@finish{nil}
5597 </nil>

```

## 18 Support for Plain T<sub>E</sub>X (plain.def)

### 18.1 Not renaming hyphen.tex

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based T<sub>E</sub>X-format. When asked he responded:

That file name is “sacred”, and if anybody changes it they will cause severe upward/downward compatibility headaches.

People can have a file `localhyphen.tex` or whatever they like, but they mustn’t diddle with `hyphen.tex` (or `plain.tex` except to preload additional fonts).

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to achieve the desired effect, based on the `babel` package. If you load each of them with `iniTEX`, you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`. As these files are going to be read as the first thing `iniTEX` sees, we need to set some category codes just to be able to change the definition of `\input`

```
5598 (*bplain | blplain)
5599 \catcode`\{=1 % left brace is begin-group character
5600 \catcode`\}=2 % right brace is end-group character
5601 \catcode`\#=6 % hash mark is macro parameter character
```

Now let’s see if a file called `hyphen.cfg` can be found somewhere on T<sub>E</sub>X’s input path by trying to open it for reading...

```
5602 \openin 0 hyphen.cfg
```

If the file wasn’t found the following test turns out true.

```
5603 \ifeof0
5604 \else
```

When `hyphen.cfg` could be opened we make sure that *it* will be read instead of the file `hyphen.tex` which should (according to Don Knuth’s ruling) contain the american English hyphenation patterns and nothing else.

We do this by first saving the original meaning of `\input` (and I use a one letter control sequence for that so as not to waste multi-letter control sequence on this in the format).

```
5605 \let\input
```

Then `\input` is defined to forget about its argument and load `hyphen.cfg` instead.

```
5606 \def\input #1 {%
5607   \let\input\input
5608   \input{#1}
5609 }
```

Once that’s done the original meaning of `\input` can be restored and the definition of `\input` can be forgotten.

```
5609 \let\input\input
5610 }
5611 \fi
5612 </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`.

```
5613 <bplain>\input plain.tex
5614 <blplain>\input blplain.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.

```
5615 \bplain\def\fmtname{babel-plain}
5616 \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.

## 18.2 Emulating some $\text{\LaTeX}$ features

The following code duplicates or emulates parts of  $\text{\LaTeX} 2_{\epsilon}$  that are needed for `babel`.

```
5617 (*plain)
5618 \def\@empty{}
5619 \def\loadlocalcfg#1{%
5620   \openin0#1.cfg
5621   \ifeof0
5622     \closein0
5623   \else
5624     \closein0
5625     {\immediate\write16{*****}%
5626      \immediate\write16{* Local config file #1.cfg used}%
5627      \immediate\write16{*}%
5628     }
5629     \input #1.cfg\relax
5630   \fi
5631   \@endoflfd}
```

## 18.3 General tools

A number of  $\text{\LaTeX}$  macro's that are needed later on.

```
5632 \long\def\@firstofone#1{#1}
5633 \long\def\@firstoftwo#1#2{#1}
5634 \long\def\@secondoftwo#1#2{#2}
5635 \def\@nnil{\@nil}
5636 \def\@gobbletwo#1#2{}
5637 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
5638 \def\@star@or@long#1{%
5639   \@ifstar
5640   {\let\l@ngrel@x\relax#1}%
5641   {\let\l@ngrel@x\long#1}}
5642 \let\l@ngrel@x\relax
5643 \def\@car#1#2\@nil{#1}
5644 \def\@cdr#1#2\@nil{#2}
5645 \let\@typeset@protect\relax
5646 \let\protected@edef\edef
5647 \long\def\@gobble#1{}
5648 \edef\@backslashchar{\expandafter\@gobble\string\}
5649 \def\strip@prefix#1>{}
5650 \def\g@addto@macro#1#2{%
5651   \toks@\expandafter{#1#2}%
5652   \xdef#1{\the\toks@}}
5653 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
5654 \def\@nameuse#1{\csname #1\endcsname}
5655 \def\@ifundefined#1{%
5656   \expandafter\ifx\csname#1\endcsname\relax
5657     \expandafter\@firstoftwo
5658   \else
```

```

5659 \expandafter\@secondoftwo
5660 \fi}
5661 \def\@expandtwoargs#1#2#3{%
5662 \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
5663 \def\zap@space#1 #2{%
5664 #1%
5665 \ifx#2\@empty\else\expandafter\zap@space\fi
5666 #2}

```

$\LaTeX_2\epsilon$  has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

5667 \ifx\@preamblecmds\@undefined
5668 \def\@preamblecmds{}
5669 \fi
5670 \def\@onlypreamble#1{%
5671 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
5672 \@preamblecmds\do#1}}
5673 \@onlypreamble\@onlypreamble

```

Mimick  $\LaTeX$ 's `\AtBeginDocument`; for this to work the user needs to add `\begindocument` to his file.

```

5674 \def\begindocument{%
5675 \@begindocumenthook
5676 \global\let\@begindocumenthook\@undefined
5677 \def\do##1{\global\let##1\@undefined}%
5678 \@preamblecmds
5679 \global\let\do\noexpand}

5680 \ifx\@begindocumenthook\@undefined
5681 \def\@begindocumenthook{}
5682 \fi
5683 \@onlypreamble\@begindocumenthook
5684 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

We also have to mimick  $\LaTeX$ 's `\AtEndOfPackage`. Our replacement macro is much simpler; it stores its argument in `\@endoflfd`.

```

5685 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
5686 \@onlypreamble\AtEndOfPackage
5687 \def\@endoflfd{}
5688 \@onlypreamble\@endoflfd
5689 \let\bbl@afterlang\@empty
5690 \chardef\bbl@opt@hyphenmap\z@

```

$\LaTeX$  needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default.

```

5691 \ifx\if@files\@undefined
5692 \expandafter\let\csname if@files\expandafter\endcsname
5693 \csname iffalse\endcsname
5694 \fi

```

Mimick  $\LaTeX$ 's commands to define control sequences.

```

5695 \def\newcommand{\@star@or@long\new@command}
5696 \def\new@command#1{%
5697 \@testopt{\@newcommand#1}0}
5698 \def\@newcommand#1[#2]{%
5699 \@ifnextchar [{\@xargdef#1[#2]}%
5700 {\@argdef#1[#2]}}
5701 \long\def\@argdef#1[#2]#3{%
5702 \@yargdef#1\@ne{#2}{#3}}

```

```

5703 \long\def\xargdef#1[#2][#3]#4{%
5704   \expandafter\def\expandafter#1\expandafter{%
5705     \expandafter\@protected@testopt\expandafter #1%
5706     \csname\string#1\expandafter\endcsname{#3}}}%
5707   \expandafter\@yargdef \csname\string#1\endcsname
5708   \tw@{#2}{#4}}
5709 \long\def\@yargdef#1#2#3{%
5710   \@tempcnta#3\relax
5711   \advance \@tempcnta \@ne
5712   \let\@hash@\relax
5713   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
5714   \@tempcntb #2%
5715   \@whilenum\@tempcntb <\@tempcnta
5716   \do{%
5717     \edef\reserved@a{\reserved@a\@hash@the\@tempcntb}%
5718     \advance\@tempcntb \@ne}%
5719   \let\@hash@###
5720   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
5721 \def\providecommand{\@star@or@long\provide@command}
5722 \def\provide@command#1{%
5723   \begingroup
5724     \escapechar\m@ne\xdef\@gtempa{\string#1}}%
5725   \endgroup
5726   \expandafter\@ifundefined\@gtempa
5727     {\def\reserved@a{\new@command#1}}%
5728     {\let\reserved@a\relax
5729     \def\reserved@a{\new@command\reserved@a}}%
5730   \reserved@a}%
5731 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
5732 \def\declare@robustcommand#1{%
5733   \edef\reserved@a{\string#1}%
5734   \def\reserved@b{#1}%
5735   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
5736   \edef#1{%
5737     \ifx\reserved@a\reserved@b
5738       \noexpand\x@protect
5739       \noexpand#1%
5740     \fi
5741     \noexpand\protect
5742     \expandafter\noexpand\csname
5743       \expandafter\@gobble\string#1 \endcsname
5744   }%
5745   \expandafter\new@command\csname
5746     \expandafter\@gobble\string#1 \endcsname
5747 }
5748 \def\x@protect#1{%
5749   \ifx\protect\@typeset@protect\else
5750     \@x@protect#1%
5751   \fi
5752 }
5753 \def\@x@protect#1\fi#2#3{%
5754   \fi\protect#1%
5755 }

```

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`.

```

5756 \def\bb1@tempa{\csname newif\endcsname\ifin@}
5757 \ifx\in@\undefined
5758   \def\in@#1#2{%
5759     \def\in@@##1##2##3\in@{%
5760       \ifx\in@@##2\in@false\else\in@true\fi}%
5761     \in@@##1\in@\in@@}
5762 \else
5763   \let\bb1@tempa\@empty
5764 \fi
5765 \bb1@tempa

```

$\LaTeX$  has a macro to check whether a certain package was loaded with specific options. The command has two extra arguments which are code to be executed in either the true or false case. This is used to detect whether the document needs one of the accents to be activated (activegrave and activeacute). For plain  $\TeX$  we assume that the user wants them to be active by default. Therefore the only thing we do is execute the third argument (the code for the true case).

```

5766 \def\@ifpackagewith#1#2#3#4{#3}

```

The  $\LaTeX$  macro `\@ifl@aded` checks whether a file was loaded. This functionality is not needed for plain  $\TeX$  but we need the macro to be defined as a no-op.

```

5767 \def\@ifl@aded#1#2#3#4{}

```

For the following code we need to make sure that the commands `\newcommand` and `\providecommand` exist with some sensible definition. They are not fully equivalent to their  $\LaTeX 2_{\epsilon}$  versions; just enough to make things work in plain  $\TeX$  environments.

```

5768 \ifx\@tempcnta\@undefined
5769   \csname newcount\endcsname\@tempcnta\relax
5770 \fi
5771 \ifx\@tempcntb\@undefined
5772   \csname newcount\endcsname\@tempcntb\relax
5773 \fi

```

To prevent wasting two counters in  $\LaTeX 2.09$  (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```

5774 \ifx\bye\@undefined
5775   \advance\count10 by -2\relax
5776 \fi
5777 \ifx\@ifnextchar\@undefined
5778   \def\@ifnextchar#1#2#3{%
5779     \let\reserved@d=#1%
5780     \def\reserved@a{#2}\def\reserved@b{#3}%
5781     \futurelet\@let@token\@ifnch}
5782   \def\@ifnch{%
5783     \ifx\@let@token\@sptoken
5784       \let\reserved@c\@xifnch
5785     \else
5786       \ifx\@let@token\reserved@d
5787         \let\reserved@c\reserved@a
5788       \else
5789         \let\reserved@c\reserved@b
5790       \fi
5791     \fi
5792     \reserved@c}
5793   \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
5794   \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
5795 \fi
5796 \def\@testopt#1#2{%

```



```

5797 \@ifnextchar[{\#1}{\#1[\#2]}}
5798 \def\@protected@testopt#1{%
5799 \ifx\protect\@typeset@protect
5800 \expandafter\@testopt
5801 \else
5802 \@@x@protect#1%
5803 \fi}
5804 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
5805 #2\relax}\fi}
5806 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
5807 \else\expandafter\@gobble\fi{#1}}

```

## 18.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain  $\TeX$  environment.

```

5808 \def\DeclareTextCommand{%
5809 \@dec@text@cmd\providecommand
5810 }
5811 \def\ProvideTextCommand{%
5812 \@dec@text@cmd\providecommand
5813 }
5814 \def\DeclareTextSymbol#1#2#3{%
5815 \@dec@text@cmd\chardef#1{#2}#3\relax
5816 }
5817 \def\@dec@text@cmd#1#2#3{%
5818 \expandafter\def\expandafter#2%
5819 \expandafter{%
5820 \csname#3-cmd\expandafter\endcsname
5821 \expandafter#2%
5822 \csname#3\string#2\endcsname
5823 }%
5824 % \let\@ifdefinable\@rc@ifdefinable
5825 \expandafter#1\csname#3\string#2\endcsname
5826 }
5827 \def\@current@cmd#1{%
5828 \ifx\protect\@typeset@protect\else
5829 \noexpand#1\expandafter\@gobble
5830 \fi
5831 }
5832 \def\@changed@cmd#1#2{%
5833 \ifx\protect\@typeset@protect
5834 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
5835 \expandafter\ifx\csname ?\string#1\endcsname\relax
5836 \expandafter\def\csname ?\string#1\endcsname{%
5837 \@changed@x@err{#1}%
5838 }%
5839 \fi
5840 \global\expandafter\let
5841 \csname\cf@encoding\string#1\expandafter\endcsname
5842 \csname ?\string#1\endcsname
5843 \fi
5844 \csname\cf@encoding\string#1%
5845 \expandafter\endcsname
5846 \else
5847 \noexpand#1%
5848 \fi
5849 }
5850 \def\@changed@x@err#1{%

```

```

5851 \errhelp{Your command will be ignored, type <return> to proceed}%
5852 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
5853 \def\DeclareTextCommandDefault#1{%
5854 \DeclareTextCommand#1?%
5855 }
5856 \def\ProvideTextCommandDefault#1{%
5857 \ProvideTextCommand#1?%
5858 }
5859 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
5860 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
5861 \def\DeclareTextAccent#1#2#3{%
5862 \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
5863 }
5864 \def\DeclareTextCompositeCommand#1#2#3#4{%
5865 \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
5866 \edef\reserved@b{\string##1}%
5867 \edef\reserved@c{%
5868 \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
5869 \ifx\reserved@b\reserved@c
5870 \expandafter\expandafter\expandafter\ifx
5871 \expandafter\@car\reserved@a\relax\relax\@nil
5872 \@text@composite
5873 \else
5874 \edef\reserved@b##1{%
5875 \def\expandafter\noexpand
5876 \csname#2\string#1\endcsname####1{%
5877 \noexpand\@text@composite
5878 \expandafter\noexpand\csname#2\string#1\endcsname
5879 ####1\noexpand\@empty\noexpand\@text@composite
5880 {##1}%
5881 }%
5882 }%
5883 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
5884 \fi
5885 \expandafter\def\csname\expandafter\string\csname
5886 #2\endcsname\string#1-\string#3\endcsname{#4}
5887 \else
5888 \errhelp{Your command will be ignored, type <return> to proceed}%
5889 \errmessage{\string\DeclareTextCompositeCommand\space used on
5890 inappropriate command \protect#1}
5891 \fi
5892 }
5893 \def\@text@composite#1#2#3\@text@composite{%
5894 \expandafter\@text@composite@x
5895 \csname\string#1-\string#2\endcsname
5896 }
5897 \def\@text@composite@x#1#2{%
5898 \ifx#1\relax
5899 #2%
5900 \else
5901 #1%
5902 \fi
5903 }
5904 %
5905 \def\@strip@args#1:#2-#3\@strip@args{#2}
5906 \def\DeclareTextComposite#1#2#3#4{%
5907 \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
5908 \bgroup
5909 \lcode`\@=#4%

```

```

5910 \lowercase{%
5911 \egroup
5912 \reserved@a @%
5913 }%
5914 }
5915 %
5916 \def\UseTextSymbol#1#2{%
5917 % \let\@curr@enc\cf@encoding
5918 % \@use@text@encoding{#1}%
5919 #2%
5920 % \@use@text@encoding\@curr@enc
5921 }
5922 \def\UseTextAccent#1#2#3{%
5923 % \let\@curr@enc\cf@encoding
5924 % \@use@text@encoding{#1}%
5925 % #2{\@use@text@encoding\@curr@enc\selectfont#3}%
5926 % \@use@text@encoding\@curr@enc
5927 }
5928 \def\@use@text@encoding#1{%
5929 % \edef\font@encoding{#1}%
5930 % \xdef\font@name{%
5931 % \csname\curr@fontshape/\font@size\endcsname
5932 % }%
5933 % \pickup@font
5934 % \font@name
5935 % \@enc@update
5936 }
5937 \def\DeclareTextSymbolDefault#1#2{%
5938 % \DeclareTextCommandDefault#1{\UseTextSymbol{#2}{#1}}%
5939 }
5940 \def\DeclareTextAccentDefault#1#2{%
5941 % \DeclareTextCommandDefault#1{\UseTextAccent{#2}{#1}}%
5942 }
5943 \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.

```

5944 \DeclareTextAccent{"}{OT1}{127}
5945 \DeclareTextAccent{'}{OT1}{19}
5946 \DeclareTextAccent{^}{OT1}{94}
5947 \DeclareTextAccent{`}{OT1}{18}
5948 \DeclareTextAccent{~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for PLAIN  $\text{\TeX}$ .

```

5949 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
5950 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
5951 \DeclareTextSymbol{\textquoteleft}{OT1}{`\'}
5952 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
5953 \DeclareTextSymbol{\i}{OT1}{16}
5954 \DeclareTextSymbol{\ss}{OT1}{25}

```

For a couple of languages we need the  $\text{\LaTeX}$ -control sequence `\scriptsize` to be available. Because plain  $\text{\TeX}$  doesn't have such a sophisticated font mechanism as  $\text{\LaTeX}$  has, we just `\let` it to `\sevenrm`.

```

5955 \ifx\scriptsize\undefined
5956 % \let\scriptsize\sevenrm
5957 \fi
5958 </plain>

```

## 19 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).