

# Babel

Version 3.44.2025  
2020/06/01

*Original author*  
Johannes L. Braams

*Current maintainer*  
Javier Bezos

Localization and  
internationalization

Unicode

T<sub>E</sub>X

pdfT<sub>E</sub>X

LuaT<sub>E</sub>X

XeT<sub>E</sub>X

# Contents

<b>I</b>	<b>User guide</b>	<b>4</b>
<b>1</b>	<b>The user interface</b>	<b>4</b>
1.1	Monolingual documents . . . . .	4
1.2	Multilingual documents . . . . .	6
1.3	Mostly monolingual documents . . . . .	7
1.4	Modifiers . . . . .	8
1.5	Troubleshooting . . . . .	8
1.6	Plain . . . . .	9
1.7	Basic language selectors . . . . .	9
1.8	Auxiliary language selectors . . . . .	10
1.9	More on selection . . . . .	11
1.10	Shorthands . . . . .	12
1.11	Package options . . . . .	15
1.12	The base option . . . . .	17
1.13	ini files . . . . .	18
1.14	Selecting fonts . . . . .	26
1.15	Modifying a language . . . . .	28
1.16	Creating a language . . . . .	29
1.17	Digits and counters . . . . .	32
1.18	Accessing language info . . . . .	33
1.19	Hyphenation and line breaking . . . . .	34
1.20	Selection based on BCP 47 tags . . . . .	37
1.21	Selecting scripts . . . . .	38
1.22	Selecting directions . . . . .	38
1.23	Language attributes . . . . .	43
1.24	Hooks . . . . .	43
1.25	Languages supported by babel with ldf files . . . . .	44
1.26	Unicode character properties in luatex . . . . .	45
1.27	Tweaking some features . . . . .	46
1.28	Tips, workarounds, known issues and notes . . . . .	46
1.29	Current and future work . . . . .	47
1.30	Tentative and experimental code . . . . .	48
<b>2</b>	<b>Loading languages with language.dat</b>	<b>48</b>
2.1	Format . . . . .	48
<b>3</b>	<b>The interface between the core of babel and the language definition files</b>	<b>49</b>
3.1	Guidelines for contributed languages . . . . .	50
3.2	Basic macros . . . . .	51
3.3	Skeleton . . . . .	52
3.4	Support for active characters . . . . .	53
3.5	Support for saving macro definitions . . . . .	53
3.6	Support for extending macros . . . . .	54
3.7	Macros common to a number of languages . . . . .	54
3.8	Encoding-dependent strings . . . . .	54
<b>4</b>	<b>Changes</b>	<b>58</b>
4.1	Changes in babel version 3.9 . . . . .	58
<b>II</b>	<b>Source code</b>	<b>59</b>

<b>5</b>	<b>Identification and loading of required files</b>	<b>59</b>
<b>6</b>	<b>locale directory</b>	<b>59</b>
<b>7</b>	<b>Tools</b>	<b>60</b>
7.1	Multiple languages . . . . .	64
7.2	The Package File ( $\LaTeX$ , babel.sty) . . . . .	64
7.3	base . . . . .	66
7.4	Conditional loading of shorthands . . . . .	68
7.5	Cross referencing macros . . . . .	69
7.6	Marks . . . . .	72
7.7	Preventing clashes with other packages . . . . .	73
7.7.1	ifthen . . . . .	73
7.7.2	varioref . . . . .	74
7.7.3	hhline . . . . .	74
7.7.4	hyperref . . . . .	75
7.7.5	fancyhdr . . . . .	75
7.8	Encoding and fonts . . . . .	75
7.9	Basic bidi support . . . . .	77
7.10	Local Language Configuration . . . . .	82
<b>8</b>	<b>The kernel of Babel (babel.def, common)</b>	<b>86</b>
8.1	Tools . . . . .	86
<b>9</b>	<b>Multiple languages</b>	<b>87</b>
9.1	Selecting the language . . . . .	89
9.2	Errors . . . . .	98
9.3	Hooks . . . . .	101
9.4	Setting up language files . . . . .	103
9.5	Shorthands . . . . .	105
9.6	Language attributes . . . . .	114
9.7	Support for saving macro definitions . . . . .	116
9.8	Short tags . . . . .	117
9.9	Hyphens . . . . .	118
9.10	Multiencoding strings . . . . .	119
9.11	Macros common to a number of languages . . . . .	125
9.12	Making glyphs available . . . . .	125
9.12.1	Quotation marks . . . . .	125
9.12.2	Letters . . . . .	127
9.12.3	Shorthands for quotation marks . . . . .	128
9.12.4	Umlauts and tremas . . . . .	129
9.13	Layout . . . . .	130
9.14	Load engine specific macros . . . . .	131
9.15	Creating and modifying languages . . . . .	131
<b>10</b>	<b>Adjusting the Babel bahavior</b>	<b>147</b>
<b>11</b>	<b>Loading hyphenation patterns</b>	<b>149</b>
<b>12</b>	<b>Font handling with fontspec</b>	<b>154</b>

<b>13</b>	<b>Hooks for XeTeX and LuaTeX</b>	<b>158</b>
13.1	XeTeX . . . . .	158
13.2	Layout . . . . .	160
13.3	LuaTeX . . . . .	161
13.4	Southeast Asian scripts . . . . .	167
13.5	CJK line breaking . . . . .	171
13.6	Automatic fonts and ids switching . . . . .	171
13.7	Layout . . . . .	182
13.8	Auto bidi with basic and basic-r . . . . .	184
<b>14</b>	<b>Data for CJK</b>	<b>195</b>
<b>15</b>	<b>The ‘nil’ language</b>	<b>195</b>
<b>16</b>	<b>Support for Plain T<sub>E</sub>X (plain.def)</b>	<b>196</b>
16.1	Not renaming hyphen.tex . . . . .	196
16.2	Emulating some L <sup>A</sup> T <sub>E</sub> X features . . . . .	197
16.3	General tools . . . . .	197
16.4	Encoding related macros . . . . .	201
<b>17</b>	<b>Acknowledgements</b>	<b>204</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 . . . . .	28

## Part I

# User guide

- This user guide focuses on internationalization and localization with  $\LaTeX$ . There are also some notes on its use with Plain  $\TeX$ .
- Changes and new features with relation to version 3.8 are highlighted with **New X.XX**, and there are some notes for the latest versions in the babel wiki. The most recent features could be still unstable. Please, report any issues you find in GitHub, which is better than just complaining on an e-mail list or a web forum.
- If you are interested in the  $\TeX$  multilingual support, please join the kadingira mail list. You can follow the development of babel in GitHub (which provides many sample files, too). If you are the author of a package, feel free to send to me a few test files which I'll add to mine, so that possible issues could be caught in the development phase.
- See section 3.1 for contributing a language.
- The first sections describe the traditional way of loading a language (with `ldf` files). The alternative way based on `ini` files, which complements the previous one (it does *not* replace it), is described below.

## 1 The user interface

### 1.1 Monolingual documents

In most cases, a single language is required, and then all you need in  $\LaTeX$  is to load the package using its standard mechanism for this purpose, namely, passing that language as an optional argument. In addition, you may want to set the font and input encodings. Many languages are compatible with xetex and luatex. With them you can use babel to localize the documents. When these engines are used, the Latin script is covered by default in current  $\LaTeX$  (provided the document encoding is UTF-8), because the font loader is preloaded and the font is switched to `lmroman`. Other scripts require loading `fontspec`. You may want to set the font attributes with `fontspec`, too.

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

PDFTEX

```
\documentclass{article}

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

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

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

LUATEX/XETEX

```
\documentclass{article}

\usepackage[russian]{babel}

\babelfont{rm}{DejaVu Serif}

\begin{document}

Россия, находящаяся на пересечении множества культур, а также
с учётом многонационального характера её населения, — отличается
высокой степенью этнокультурного многообразия и способностью к
межкультурному диалогу.

\end{document}
```

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

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

Or the more explanatory:

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

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

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

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

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

**NOTE** Because of the way babel has evolved, “language” can refer to (1) a set of hyphenation patterns as preloaded into the format, (2) a package option, (3) an `ldf` file, and (4) a name used in the document to select a language or dialect. So, a package option refers to a language in a generic way – sometimes it is the actual language name used to select it, sometimes it is a file name loading a language with a different name, sometimes it is a file name loading several languages. Please, read the documentation for specific languages for further info.

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

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

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

**NOTE** With hyperref you may want to set the document language with something like:

```
\usepackage[pdflang=es-MX]{hyperref}
```

This is not currently done by babel and you must set it by hand.

## 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 `\babel font`, if used.)

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

**EXAMPLE** A trivial document is:

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
```

<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”.

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

## 1.6 Plain

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

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

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

## 1.7 Basic language selectors

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

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

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

```
\selectlanguage{german}
```

This command can be used as environment, too.

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

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

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

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

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

`\foreignlanguage` [*<option-list>*]{*<language>*}{*<text>*}

The command `\foreignlanguage` takes two arguments; the second argument is a phrase to be typeset according to the rules of the language named in its first one.

This command (1) only switches the extra definitions and the hyphenation rules for the language, *not* the names and dates, (2) does not send information about the language to auxiliary files (i.e., the surrounding language is still in force), and (3) it works even if the language has not been set as package option (but in such a case it only sets the hyphenation patterns and a warning is shown). With the `bidir` option, it also enters in horizontal mode (this is not done always for backwards compatibility).

**New 3.44** As already said, captions and dates are not switched. However, with the optional argument you can switch them, too. So, you can write:

```
\foreignlanguage[date]{polish}{\today}
```

In addition, captions can be switched with `captions` (or both, of course, with `date, captions`). Until 3.43 you had to write something like `{\selectlanguage{...} ...}`, which was not always the most convenient way.

## 1.8 Auxiliary language selectors

`\begin{otherlanguage}` {*<language>*} ... `\end{otherlanguage}`

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

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

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

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

Spaces after the environment are ignored.

`\begin{otherlanguage*}` [*<option-list>*]{*<language>*} ... `\end{otherlanguage*}`

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

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

`\begin{hyphenrules}` {*<language>*} ... `\end{hyphenrules}`

The environment `hyphenrules` can be used to select *only* the hyphenation rules to be used (it can be used as command, too). This can for instance be used to select ‘nohyphenation’,

---

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

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 other `language*` (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

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

you can write

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

and

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

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

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

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

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

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

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

```
\babelensure{polish}
```

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

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

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

## 1.10 Shorthands

A *shorthand* is a sequence of one or two characters that expands to arbitrary  $\TeX$  code. Shorthands can be used for different kinds of things, for example: (1) in some languages shorthands such as "a are defined to be able to hyphenate the word if the encoding is OT1; (2) in some languages shorthands such as ! are used to insert the right amount of white space; (3) several kinds of discretionary and breaks can be inserted easily with "-", "=", etc. The package `inputenc` as well as `xetex` and `luatex` have alleviated entering non-ASCII characters, but minority languages and some kinds of text can still require characters not directly available on the keyboards (and sometimes not even as separated or precomposed Unicode characters). As to the point 2, now `pdfTeX` provides `\kernbcode`, and `luatex` can manipulate the glyph list. Tools for point 3 can be still very useful in general. There are three levels of shorthands: *user*, *language*, and *system* (by order of precedence). Version 3.9 introduces the *language user* level on top of the user level, as described below. In most cases, you will use only shorthands provided by languages.

**NOTE** Note the following:

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

**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.

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

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

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

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

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

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

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

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

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

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

However, 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:

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

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

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

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

**`\languageshorthands`** `{⟨language⟩}`

The command `\languageshorthands` can be used to switch the shorthands on the language level. It takes one argument, the name of a language or none (the latter does what its name suggests).<sup>6</sup> Note that for this to work the language should have been specified as an option when loading the babel package. For example, you can use in english the shorthands defined by ngerman with

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

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

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

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

**`\babelshorthand`** `{⟨shorthand⟩}`

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

<sup>6</sup>Actually, any name not corresponding to a language group does the same as none. However, follow this convention because it might be enforced in future releases of babel to catch possible errors.

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

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

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

In addition, the babel core declares ~ as a one-char shorthand which is let, like the standard ~, to a non breaking space.<sup>8</sup>

`\ifbabelshorthand` {<character>}{<true>}{<false>}

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

`\aliasshorthand` {<original>}{<alias>}

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

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

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



<b>KeepShorthandsActive</b>	Tells babel not to deactivate shorthands after loading a language file, so that they are also available in the preamble.
<b>activeacute</b>	For some languages babel supports this options to set ' as a shorthand in case it is not done by default.
<b>activegrave</b>	Same for `.
<b>shorthands=</b>	<p><math>\langle char \rangle \langle char \rangle \dots</math>   off</p> <p>The only language shorthands activated are those given, like, eg:</p> <pre>\usepackage[esperanto,french,shorthands=;!]{babel}</pre> <p>If ' is included, activeacute is set; if ` is included, activegrave is set. Active characters (like ~) should be preceded by \string (otherwise they will be expanded by <math>\LaTeX</math> before they are passed to the package and therefore they will not be recognized); however, t is provided for the common case of ~ (as well as c for not so common case of the comma). With shorthands=off no language shorthands are defined, As some languages use this mechanism for tools not available otherwise, a macro \babelshorthand is defined, which allows using them; see above.</p>
<b>safe=</b>	<p>none   ref   bib</p> <p>Some <math>\LaTeX</math> macros are redefined so that using shorthands is safe. With safe=bib only \nocite, \bibtex and \bibitem are redefined. With safe=ref only \newlabel, \ref and \pageref are redefined (as well as a few macros from varioref and ifthen). With safe=none no macro is redefined. This option is strongly recommended, because a good deal of incompatibilities and errors are related to these redefinitions. As of <b>New 3.34</b>, in <math>\epsilon\TeX</math> based engines (ie, almost every engine except the oldest ones) shorthands can be used in these macros (formerly you could not).</p>
<b>math=</b>	<p>active   normal</p> <p>Shorthands are mainly intended for text, not for math. By setting this option with the value normal they are deactivated in math mode (default is active) and things like <math>\{a'\}</math> (a closing brace after a shorthand) are not a source of trouble anymore.</p>
<b>config=</b>	<p><math>\langle file \rangle</math></p> <p>Load <math>\langle file \rangle.cfg</math> instead of the default config file bblopts.cfg (the file is loaded even with noconfigs).</p>
<b>main=</b>	<p><math>\langle language \rangle</math></p> <p>Sets the main language, as explained above, ie, this language is always loaded last. If it is not given as package or global option, it is added to the list of requested languages.</p>
<b>headfoot=</b>	<p><math>\langle language \rangle</math></p> <p>By default, headlines and footlines are not touched (only marks), and if they contain language-dependent macros (which is not usual) there may be unexpected results. With this option you may set the language in heads and foots.</p>

- noconfigs** Global and language default config files are not loaded, so you can make sure your document is not spoiled by an unexpected .cfg file. However, if the key config is set, this file is loaded.
- showlanguages** Prints to the log the list of languages loaded when the format was created: number (remember dialects can share it), name, hyphenation file and exceptions file.
- nocase** **New 3.9l** Language settings for uppercase and lowercase mapping (as set by \SetCase) are ignored. Use only if there are incompatibilities with other packages.
- silent** **New 3.9l** No warnings and no *infos* are written to the log file.<sup>9</sup>
- strings=** generic | unicode | encoded | *<label>* | *<font encoding>*  
 Selects the encoding of strings in languages supporting this feature. Predefined labels are generic (for traditional T<sub>E</sub>X, LICR and ASCII strings), unicode (for engines like xetex and luatex) and encoded (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in \MakeUppercase and the like (this feature misuses some internal L<sup>A</sup>T<sub>E</sub>X tools, so use it only as a last resort).
- hyphenmap=** off | first | select | other | other\*  
**New 3.9g** Sets the behavior of case mapping for hyphenation, provided the language defines it.<sup>10</sup> It can take the following values:  
**off** deactivates this feature and no case mapping is applied;  
**first** sets it at the first switching commands in the current or parent scope (typically, when the aux file is first read and at \begin{document}, but also the first \selectlanguage in the preamble), and it's the default if a single language option has been stated;<sup>11</sup>  
**select** sets it only at \selectlanguage;  
**other** also sets it at otherlanguage;  
**other\*** also sets it at otherlanguage\* as well as in heads and foots (if the option headfoot is used) and in auxiliary files (ie, at \select@language), and it's the default if several language options have been stated. The option first can be regarded as an optimized version of other\* for monolingual documents.<sup>12</sup>
- bidi=** default | basic | basic-r | bidi-l | bidi-r  
**New 3.14** Selects the bidi algorithm to be used in luatex and xetex. See sec. 1.22.
- layout=** **New 3.16** Selects which layout elements are adapted in bidi documents. See sec. 1.22.

## 1.12 The base option

With this package option babel just loads some basic macros (those in switch.def), defines \AfterBabelLanguage and exits. It also selects the hyphenation patterns for the

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

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

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

<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.

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

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

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

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

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

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

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

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

### 1.13 ini files

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

`ini` files are not meant only for `babel`, and they have been devised as a resource for other packages. To easy interoperability between  $\TeX$  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 `\...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, and as alternative when the `ldf`, for some reason, does work as expected.

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

LUATEX/XETEX

```
\documentclass{book}

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

```

\babelfont{rm}{DejaVu Sans}

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}

```

**NOTE** The ini files just define and set some parameters, but the corresponding behavior is not always implemented. Also, there are some limitations in the engines. A few remarks follow (which could no longer be valid when you read this manual, if the packages involved have been updated):

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

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

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

```
\newfontscript{Devanagari}{deva}
```

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

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

```

\babelprovide[import,hyphenrules=+]{lao}
\babelpatterns[lao]{1ᦅ 1ᦆ 1ᦇ 1ᦈ 1ᦉ 1ᦊ} % Random

```

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

```

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

```

**Latin, Greek, Cyrillic** Combining chars with the default luatex font renderer might be wrong; on the other hand, with the Harfbuzz renderer diacritics are stacked

correctly, but many hyphenations points are discarded (this bug seems related to kerning, so it depends on the font). Fortunately, fonts can be loaded twice with different renderers; for example:

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

With xetex both combining characters and hyphenation work as expected.

**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>	cu	Church Slavic
agq	Aghem	cu-Cyrs	Church Slavic
ak	Akan	cu-Glag	Church Slavic
am	Amharic <sup>ul</sup>	cy	Welsh <sup>ul</sup>
ar	Arabic <sup>ul</sup>	da	Danish <sup>ul</sup>
ar-DZ	Arabic <sup>ul</sup>	dav	Taita
ar-MA	Arabic <sup>ul</sup>	de-AT	German <sup>ul</sup>
ar-SY	Arabic <sup>ul</sup>	de-CH	German <sup>ul</sup>
as	Assamese	de	German <sup>ul</sup>
asa	Asu	dje	Zarma
ast	Asturian <sup>ul</sup>	dsb	Lower Sorbian <sup>ul</sup>
az-Cyrl	Azerbaijani	dua	Duala
az-Latn	Azerbaijani	dyo	Jola-Fonyi
az	Azerbaijani <sup>ul</sup>	dz	Dzongkha
bas	Basaa	ebu	Embu
be	Belarusian <sup>ul</sup>	ee	Ewe
bem	Bemba	el	Greek <sup>ul</sup>
bez	Bena	el-polyton	Polytonic Greek <sup>ul</sup>
bg	Bulgarian <sup>ul</sup>	en-AU	English <sup>ul</sup>
bm	Bambara	en-CA	English <sup>ul</sup>
bn	Bangla <sup>ul</sup>	en-GB	English <sup>ul</sup>
bo	Tibetan <sup>u</sup>	en-NZ	English <sup>ul</sup>
brx	Bodo	en-US	English <sup>ul</sup>
bs-Cyrl	Bosnian	en	English <sup>ul</sup>
bs-Latn	Bosnian <sup>ul</sup>	eo	Esperanto <sup>ul</sup>
bs	Bosnian <sup>ul</sup>	es-MX	Spanish <sup>ul</sup>
ca	Catalan <sup>ul</sup>	es	Spanish <sup>ul</sup>
ce	Chechen	et	Estonian <sup>ul</sup>
cgg	Chiga	eu	Basque <sup>ul</sup>
chr	Cherokee	ewo	Ewondo
ckb	Central Kurdish	fa	Persian <sup>ul</sup>
cop	Coptic	ff	Fulah
cs	Czech <sup>ul</sup>	fi	Finnish <sup>ul</sup>

fil	Filipino	ksh	Colognian
fo	Faroese	kw	Cornish
fr	French <sup>ul</sup>	ky	Kyrgyz
fr-BE	French <sup>ul</sup>	lag	Langi
fr-CA	French <sup>ul</sup>	lb	Luxembourgish
fr-CH	French <sup>ul</sup>	lg	Ganda
fr-LU	French <sup>ul</sup>	lkt	Lakota
fur	Friulian <sup>ul</sup>	ln	Lingala
fy	Western Frisian	lo	Lao <sup>ul</sup>
ga	Irish <sup>ul</sup>	lrc	Northern Luri
gd	Scottish Gaelic <sup>ul</sup>	lt	Lithuanian <sup>ul</sup>
gl	Galician <sup>ul</sup>	lu	Luba-Katanga
grc	Ancient Greek <sup>ul</sup>	luo	Luo
gsw	Swiss German	luy	Luyia
gu	Gujarati	lv	Latvian <sup>ul</sup>
guz	Gusii	mas	Masai
gv	Manx	mer	Meru
ha-GH	Hausa	mfe	Morisyen
ha-NE	Hausa <sup>l</sup>	mg	Malagasy
ha	Hausa	mgf	Makhuwa-Meetto
haw	Hawaiian	mgh	Meta'
he	Hebrew <sup>ul</sup>	mgo	Meta'
hi	Hindi <sup>u</sup>	mk	Macedonian <sup>ul</sup>
hr	Croatian <sup>ul</sup>	ml	Malayalam <sup>ul</sup>
hsb	Upper Sorbian <sup>ul</sup>	mn	Mongolian
hu	Hungarian <sup>ul</sup>	mr	Marathi <sup>ul</sup>
hy	Armenian <sup>u</sup>	ms-BN	Malay <sup>l</sup>
ia	Interlingua <sup>ul</sup>	ms-SG	Malay <sup>l</sup>
id	Indonesian <sup>ul</sup>	ms	Malay <sup>ul</sup>
ig	Igbo	mt	Maltese
ii	Sichuan Yi	mua	Mundang
is	Icelandic <sup>ul</sup>	my	Burmese
it	Italian <sup>ul</sup>	mzn	Mazanderani
ja	Japanese	naq	Nama
jgo	Ngomba	nb	Norwegian Bokmål <sup>ul</sup>
jmc	Machame	nd	North Ndebele
ka	Georgian <sup>ul</sup>	ne	Nepali
kab	Kabyle	nl	Dutch <sup>ul</sup>
kam	Kamba	nmg	Kwasio
kde	Makonde	nn	Norwegian Nynorsk <sup>ul</sup>
kea	Kabuverdianu	nnh	Ngiemboon
khq	Koyra Chiini	nus	Nuer
ki	Kikuyu	nyn	Nyankole
kk	Kazakh	om	Oromo
kkj	Kako	or	Odia
kl	Kalaallisut	os	Ossetic
klj	Kalenjin	pa-Arab	Punjabi
km	Khmer	pa-Guru	Punjabi
kn	Kannada <sup>ul</sup>	pa	Punjabi
ko	Korean	pl	Polish <sup>ul</sup>
kok	Konkani	pms	Piedmontese <sup>ul</sup>
ks	Kashmiri	ps	Pashto
ksb	Shambala	pt-BR	Portuguese <sup>ul</sup>
ksf	Bafia	pt-PT	Portuguese <sup>ul</sup>
		pt	Portuguese <sup>ul</sup>

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

---

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

---

aghem	amharic
akan	ancientgreek
albanian	arabic
american	arabic-algeria

arabic-DZ	chinese-simplified-singapore
arabic-morocco	chinese-simplified
arabic-MA	chinese-traditional-hongkongsarchina
arabic-syria	chinese-traditional-macausarchina
arabic-SY	chinese-traditional
armenian	chinese
assamese	churchslavic
asturian	churchslavic-cyrs
asu	churchslavic-oldcyrillic <sup>13</sup>
australian	churchsslavic-glag
austrian	churchsslavic-glagolitic
azerbaijani-cyrillic	colognian
azerbaijani-cyrl	cornish
azerbaijani-latin	croatian
azerbaijani-latn	czech
azerbaijani	danish
bafia	duala
bambara	dutch
basaa	dzongkha
basque	embu
belarusian	english-au
bemba	english-australia
beni	english-ca
bengali	english-canada
bodo	english-gb
bosnian-cyrillic	english-newzealand
bosnian-cyrl	english-nz
bosnian-latin	english-unitedkingdom
bosnian-latn	english-unitedstates
bosnian	english-us
brazilian	english
breton	esperanto
british	estonian
bulgarian	ewe
burmese	ewondo
canadian	faroes
cantonese	filipino
catalan	finnish
centralatlastamazight	french-be
centralkurdish	french-belgium
chechen	french-ca
cherokee	french-canada
chiga	french-ch
chinese-hans-hk	french-lu
chinese-hans-mo	french-luxembourg
chinese-hans-sg	french-switzerland
chinese-hans	french
chinese-hant-hk	friulian
chinese-hant-mo	fulah
chinese-hant	galician
chinese-simplified-hongkongsarchina	ganda
chinese-simplified-macausarchina	georgian

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



german-at	luxembourgish
german-austria	luyia
german-ch	macedonian
german-switzerland	machame
german	makhuwameetto
greek	makonde
gujarati	malagasy
gusii	malay-bn
hausa-gh	malay-brunei
hausa-ghana	malay-sg
hausa-ne	malay-singapore
hausa-niger	malay
hausa	malayalam
hawaiian	maltese
hebrew	manx
hindi	marathi
hungarian	masai
icelandic	mazanderani
igbo	meru
inarisami	meta
indonesian	mexican
interlingua	mongolian
irish	morisyen
italian	mundang
japanese	nama
jolafonyi	nepali
kabuverdianu	newzealand
kabyle	ngiemboon
kako	ngomba
kalaallisut	norsk
kalenjin	northernluri
kamba	northernsami
kannada	northndebele
kashmiri	norwegianbokmal
kazakh	norwegiannynorsk
khmer	nswissgerman
kikuyu	nuer
kinyarwanda	nyankole
konkani	nynorsk
korean	occitan
koyraborosenni	oriya
koyrachiini	oromo
kwasio	ossetic
kyrgyz	pashto
lakota	persian
langi	piedmontese
lao	polish
latvian	polytonicgreek
lingala	portuguese-br
lithuanian	portuguese-brazil
lowersorbian	portuguese-portugal
lsorbian	portuguese-pt
lubakatanga	portuguese
luo	punjabi-arab

punjabi-arabic	soga
punjabi-gurmukhi	somali
punjabi-guru	spanish-mexico
punjabi	spanish-mx
quechua	spanish
romanian	standardmoroccantamazight
romansh	swahili
rombo	swedish
rundi	swissgerman
russian	tachelhit-latin
rwa	tachelhit-latn
sakha	tachelhit-tfng
samburu	tachelhit-tifinagh
samin	tachelhit
sango	taita
sangu	tamil
sanskrit-beng	tasawaq
sanskrit-bengali	telugu
sanskrit-deva	teso
sanskrit-devanagari	thai
sanskrit-gujarati	tibetan
sanskrit-gujr	tigrinya
sanskrit-kannada	tongan
sanskrit-knda	turkish
sanskrit-malayalam	turkmen
sanskrit-mlym	ukenglish
sanskrit-telu	ukrainian
sanskrit-telugu	upporsorbian
sanskrit	urdu
scottishgaelic	usenglish
sena	usorbian
serbian-cyrillic-bosniaherzegovina	uyghur
serbian-cyrillic-kosovo	uzbek-arab
serbian-cyrillic-montenegro	uzbek-arabic
serbian-cyrillic	uzbek-cyrillic
serbian-cyrl-ba	uzbek-cyrl
serbian-cyrl-me	uzbek-latin
serbian-cyrl-xk	uzbek-latn
serbian-cyrl	uzbek
serbian-latin-bosniaherzegovina	vai-latin
serbian-latin-kosovo	vai-latn
serbian-latin-montenegro	vai-vai
serbian-latin	vai-vaii
serbian-latn-ba	vai
serbian-latn-me	vietnam
serbian-latn-xk	vietnamese
serbian-latn	vunjo
serbian	walser
shambala	welsh
shona	westernfrisian
sichuanyi	yangben
sinhala	yiddish
slovak	yoruba
slovene	zarma
slovenian	zulu afrikaans

---

### Modifying and adding values to ini files

**New 3.39** There is a way to modify the values of ini files when they get loaded with `\babelprovide` and `import`. To set, say, `digits.native` in the `numbers` section, use something like `numbers/digits.native=abcdefghijkl`. Keys may be added, too. Without `import` you may modify the identification keys. This can be used to create private variants easily. All you need is to import the same ini file with a different locale name and different parameters.

## 1.14 Selecting fonts

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

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

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

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

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

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

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

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

LUATEX/XETEX

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עברית} svenska.

\end{document}
```

---

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

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

As of 3.15, there is no need to hide spaces with % (babel removes them), but it is advisable to do 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\extrarussian{\mymacro}
```

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

- With data import'ed from ini files, you can modify the values of specific keys, like:

```
\babelprovide[import, captions/listtable = Lista de tablas]{spanish}
```

(In this particular case, instead of the captions group you may need to modify the `captions.licr` one.)

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

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

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

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

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

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

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

## 1.16 Creating a language

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

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

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

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

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

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

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

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

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

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

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

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

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

`import=` *<language-tag>*

**New 3.13** Imports data from an ini file, including captions, date, and hyphenmins. For example:

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

Unicode engines load the UTF-8 variants, while 8-bit engines load the LICR (ie, with macros like `\'` or `\ss`) ones.

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

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

There are about 200 ini files, with data taken from the ldf files and the CLDR provided by Unicode. Not all languages in the latter are complete, and therefore neither are the ini files. A few languages will show a warning about the current lack of suitability of the date format (french, breton, and occitan).

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

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

`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 (thus overriding that set when babel is loaded). Only in newly defined languages.

**EXAMPLE** Let's assume your document is mainly in Polytonic Greek, but with some sections in Italian. Then, the first attempt should be:

```
\usepackage[italian, greek.polutonic]{babel}
```

But if, say, accents in Greek are not shown correctly, you could try:

```
\usepackage[italian]{babel}
\babelprovide[import, main]{polytonicgreek}
```

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

**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=** *<language-name>*

**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 `\defineshortand` as described above. (2) Captions and `\today` are “ensured” with `\babelensure` (this is the default in ini-based languages).

## 1.17 Digits and counters

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

For example:

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

Languages providing native digits in all or some variants are:

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

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

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

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

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

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

The styles are:

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

## 1.18 Accessing language info

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

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

`\iflanguage` `{<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`  $\{\langle field \rangle\}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`\localeid`

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

**NOTE** The `\localeid` is not the same as the `\language` identifier, which refers to a set of hyphenation patters (which, in turn, is just a component of the line breaking algorithm described in the next section). The data about preloaded patterns are store in an internal macro named `\bbl@languages` (see the code for further details), but note several locales may share a single `\language`, so they are separated concepts. In `luatex`, the `\localeid` is saved in each node (where it makes sense) as an attribute, too.

## 1.19 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 (although in a limited way), while `luatex` provides basic rules for the latter, too.

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

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

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

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

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

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

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

There are also some differences with  $\LaTeX$ : (1) the character used is that set for the current font, while in  $\LaTeX$  it is hardwired to `-` (a typical value); (2) the hyphen to be used in fonts with a negative `\hyphenchar` is `-`, like in  $\LaTeX$ , but it can be changed to another value by redefining `\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`,<sup>15</sup> adds or replaces patterns for the languages given or, without the optional argument, for *all* languages. If a pattern for a certain combination already exists, it gets replaced by the new one.

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

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

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

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

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

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

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

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

<sup>15</sup>With `luatex` exceptions and patterns can be modified almost freely. However, this is very likely a task for a separate package and babel only provides the most basic tools.

them). Thus, to enter *ž* as *zh* and *š* 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 Selection based on BCP 47 tags

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

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

Here is a minimal example:

```
\documentclass{article}

\usepackage[danish]{babel}

\babeladjust{ autload.bcp47 = on }

\begin{document}

\today

\selectlanguage{fr-CA}

\today

\end{document}
```

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

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

`autload.bcp47` with values `on` and `off`.

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

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

## 1.21 Selecting scripts

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

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

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

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

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

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

## 1.22 Selecting directions

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

**WARNING** The current code for `text` in `luatex` should be considered essentially stable, but, of course, it is not bug-free and there could be improvements in the future, because setting bidi text has many subtleties (see for example <https://www.w3.org/TR/html-bidi/>). A basic stable version for other engines must wait. This applies to `text`; there is a basic support for **graphical** elements, including the

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

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

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 Ἀραβία)، استخدم الرومان ثلاث
    بادئات بـ“Arabia” على ثلاث مناطق من شبه الجزيرة العربية، إلا أنها
    حقيقةً كانت أكبر مما تعرف عليه اليوم.

\end{document}
```

**EXAMPLE** With `bidi=basic` both L and R text can be mixed without explicit markup (the latter will be only necessary in some special cases where the Unicode algorithm fails). It is used much like `bidi=basic-r`, but with R text inside L text you may want to map the



font so that the correct features are in force. This is accomplished with an option in `\babelprovide`, as illustrated:

```
\documentclass{book}

\usepackage[english, bidirectional]{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 \textit{fuṣḥā l-ʿaṣr} (MSA) and
\textit{fuṣḥā t-turāth} (CA).

\end{document}
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
\BabelFootnote{\enfootnote}{english}\{ }{.}
```

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

## 1.23 Language attributes

### `\languageattribute`

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

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

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

## 1.24 Hooks

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

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

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

Names containing the string `babel` are reserved (they are used, for example, by `\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:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## 1.25 Languages supported by babel with ldf files

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

**Afrikaans** afrikaans

**Azerbaijani** azerbaijani

**Basque** basque

**Breton** breton

**Bulgarian** bulgarian

**Catalan** catalan

**Croatian** croatian

**Czech** czech

**Danish** danish

**Dutch** dutch

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

**Esperanto** esperanto

**Estonian** estonian

**Finnish** finnish

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

**Galician** galician

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

**Greek** greek, polutonikogreek

**Hebrew** hebrew

**Icelandic** icelandic

**Indonesian** indonesian (bahasa, indon, bahasai)

**Interlingua** interlingua

**Irish Gaelic** irish  
**Italian** italian  
**Latin** latin  
**Lower Sorbian** lowersorbian  
**Malay** malay, melayu (bahasam)  
**North Sami** samin  
**Norwegian** norsk, nynorsk  
**Polish** polish  
**Portuguese** portuguese, brazilian (portuges, brazil)<sup>19</sup>  
**Romanian** romanian  
**Russian** russian  
**Scottish Gaelic** scottish  
**Spanish** spanish  
**Slovakian** slovak  
**Slovenian** slovene  
**Swedish** swedish  
**Serbian** serbian  
**Turkish** turkish  
**Ukrainian** ukrainian  
**Upper Sorbian** uppsorbian  
**Welsh** welsh

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

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

```

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

```

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

## 1.26 Unicode character properties in luatex

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

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

**New 3.32** Here,  $\{ \langle char-code \rangle \}$  is a number (with  $\text{\TeX}$  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:

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

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

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

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

## 1.27 Tweaking some features

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

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

## 1.28 Tips, workarounds, known issues and notes

- If you use the document class *book* and you use `\ref` inside the argument of `\chapter` (or just use `\ref` inside `\MakeUppercase`),  $\text{\LaTeX}$  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  $\text{\TeX}$  only takes into account the values when the paragraph is hyphenated, i.e., when it has been

finished.<sup>20</sup> So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of  $\TeX$ , not of babel. Alternatively, you may use `\usesorthands` to activate ' and `\definesorthand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).

- `\bibitem` is out of sync with `\selectlanguage` in the `.aux` file. The reason is `\bibitem` uses `\immediate` (and others, in fact), while `\selectlanguage` doesn't. There is no known workaround.
- Babel does not take into account `\normalsfcodes` and (non-)French spacing is not always properly (un)set by languages. However, problems are unlikely to happen and therefore this part remains untouched in version 3.9 (but it is in the 'to do' list).
- Using a character mathematically active (ie, with math code "8000) as a shorthand can make  $\TeX$  enter in an infinite loop in some rare cases. (Another issue in the 'to do' list, although there is a partial solution.)

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

**csquotes** Logical markup for quotes.

**iflang** Tests correctly the current language.

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

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

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

**biblatex** Programmable bibliographies and citations.

**bicaption** Bilingual captions.

**babelbib** Multilingual bibliographies.

**microtype** Adjusts the typesetting according to some languages (kerning and spacing).

Ligatures can be disabled.

**substitutefont** Combines fonts in several encodings.

**mkpattern** Generates hyphenation patterns.

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

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

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

## 1.29 Current and future work

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

Useful additions would be, for example, time, currency, addresses and personal names.<sup>21</sup> But that is the easy part, because they don't require modifying the  $\LaTeX$  internals.

Calendars (Arabic, Persian, Indic, etc.) are under study.

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

An option to manage bidirectional document layout in 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

<sup>20</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.

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



`\specials` remain, like color and hyperlinks), so babel resorts to the bidi package (by Vafa Khalighi). See the babel repository for a small example (xe-bidi).

### 1.30 Tentative and experimental code

See the code section for `\foreignlanguage*` (a new starred version of `\foreignlanguage`). For old an deprecated functions, see the wiki.

#### `\babelprehyphenation`

**New 3.44** Note it is tentative, but the current behavior for glyphs should be correct. It is similar to `\babelposthyphenation`, but (as its name implies) applied before hyphenation. There are other differences: (1) the first argument is the locale instead the name of hyphenation patterns; (2) in the search patterns = has no special meaning (| is still reserved, but currently unused); (3) in the replacement, discretionaries are not accepted, only remove, , and string = ...  
Currently it handles glyphs, not discretionaries or spaces (in particular, it will not catch the hyphen and you can't insert or remove spaces). Also, you are limited to substitutions as done by lua, although a future implementation may alternatively accept lpeg.  
Performance is still somewhat poor.

## 2 Loading languages with `language.dat`

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

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

### 2.1 Format

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

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

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

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

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

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

```
=british

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

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

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

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

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

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

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

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

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

The following assumptions are made:

- Some of the language-specific definitions might be used by plain  $\text{T}_{\text{E}}\text{X}$  users, so the files have to be coded so that they can be read by both  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  and plain  $\text{T}_{\text{E}}\text{X}$ . The current format can be checked by looking at the value of the macro `\fmtname`.
- The common part of the `babel` system redefines a number of macros and environments (defined previously in the document style) to put in the names of macros that replace the previously hard-wired texts. These macros have to be defined in the language definition files.
- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are `\<lang>hyphenmins`, `\captions<lang>`, `\date<lang>`, `\extras<lang>` and `\noextras<lang>` (the last two may be left empty); where `<lang>` is either the name of the language definition file or the name of the  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  option that is to be used. These macros and their functions are discussed below. You must define all or none for a language (or a dialect); defining, say, `\date<lang>` but not `\captions<lang>` does not raise an error but can lead to unexpected results.

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

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

Some recommendations:

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

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

### 3.1 Guidelines for contributed languages

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

- Do not hesitate stating on the file heads you are the author and the maintainer, if you actually are. There is no need to state the babel maintainer(s) as authors if they have not contributed significantly to your language files.
- Fonts are not strictly part of a language, so they are best placed in the corresponding TeX tree. This includes not only `tfm`, `vf`, `ps1`, `otf`, `mf` files and the like, but also `fd` ones.

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

- 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. Here “language” is used in the T<sub>E</sub>X 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 T<sub>E</sub>X sense of set of hyphenation patterns.

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

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

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

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

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

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

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

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

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

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

**\ProvidesLanguage** The macro `\ProvidesLanguage` should be used to identify the language definition files. Its

	syntax is similar to the syntax of the $\LaTeX$ 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, $\LaTeX$ 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 $\LaTeX$ to use a font from the second family when a font from the first family in the given encoding seems to be needed.

### 3.3 Skeleton

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

```

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

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

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

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

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

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

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

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

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

```

```
% More strings

\EndBabelCommands

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

\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  $\text{\LaTeX}$  to give a character the category code ‘active’. When a character has been made active it will remain that way until the end of the document. Its definition may vary.

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

The command `\bbl@activate` is used to change the way an active character expands. `\bbl@activate` ‘switches on’ the active 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  $\text{\TeX}$ book states: “Plain  $\text{\TeX}$  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`.  $\text{\LaTeX}$  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>27</sup>.

`\babel@save`

To save the current meaning of any control sequence, the macro `\babel@save` is provided.

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

It takes one argument,  $\langle csname \rangle$ , the control sequence for which the meaning has to be saved.

`\babel@savevariable`

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

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

### 3.6 Support for extending macros

`\addto`

The macro `\addto{ $\langle control sequence \rangle$ { $\langle \TeX code \rangle$ }}` 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 `\set@low@box` is available. It takes one argument and puts that argument in an `\hbox`, at the baseline. The result is available in `\box0` for further processing.

`\save@sf@q`

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

`\bbl@frenchspacing`

`\bbl@nonfrenchspacing`

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

### 3.8 Encoding-dependent strings

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

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

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



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

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

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

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

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

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

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

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

\EndBabelCommands
```

A real example is:

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

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



```

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

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

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

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

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

\EndBabelCommands

```

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

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

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

`\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

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

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  $\text{\TeX}$ , 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}{}
\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**  $\{ \langle to-lower-macros \rangle \}$

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

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

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

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

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

## 4 Changes

### 4.1 Changes in babel version 3.9

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

- `\select@language` did not set `\language`. This meant the language in force when auxiliary files were loaded was the one used in, for example, shorthands – if the language was `german`, a `\select@language{spanish}` had no effect.
- `\foreignlanguage` and `otherlanguage*` messed up `\extras<language>`. Scripts, encodings and many other things were not switched correctly.
- The `:ENC` mechanism for hyphenation patterns used the encoding of the *previous* language, not that of the language being selected.
- `'` (with `activeacute`) had the original value when writing to an auxiliary file, and things like an infinite loop 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 following description is no longer valid, because switch and plain have been merged into babel.def.**

The babel package after unpacking consists of the following files:

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

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

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

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

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

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 (which start always with a lowercase case). There is an exception, however: the section counters has been devised to have arbitrary keys, so you can add lowercased keys if you want.

## 7 Tools

```
1 <<version=3.44.2025>>
2 <<date=2020/06/01>>
```

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

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

This does not hurt, but should be fixed somehow.

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

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

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

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

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

```

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##1 \@nil{\bbl@trim@i##1}}
45 \bbl@tempa{ }
46 \long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
47 \long\def\bbl@trim@def#1{\bbl@trim{\def#1}}

```

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

```

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

```

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

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

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

```
71 \def\bbl@forkv#1#2{%
72   \def\bbl@kvcmd##1##2##3{#2}%
73   \bbl@kvnext#1,\@nil,}
74 \def\bbl@kvnext#1,{%
75   \ifx\@nil#1\relax\else
76     \bbl@ifblank{#1}{\bbl@forkv@eq#1=\@empty=\@nil{#1}}%
77     \expandafter\bbl@kvnext
78   \fi}
79 \def\bbl@forkv@eq#1=#2=#3\@nil#4{%
80   \bbl@trim@def\bbl@forkv@a{#1}%
81   \bbl@trim{\expandafter\bbl@kvcmd\expandafter{\bbl@forkv@a}}{#2}{#4}}
```

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

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

`\bbl@replace`

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

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

```
103 \ifx\detokenize\undefined\else % Unused macros if old Plain TeX
104   \bbl@exp{\def\\bbl@parsedef##1\detokenize{macro:}}#2->#3\relax%
105   \def\bbl@tempa{#1}%
106   \def\bbl@tempb{#2}%
107   \def\bbl@tempe{#3}}
108 \def\bbl@sreplace#1#2#3{%
```

```

109 \begingroup
110 \expandafter\bb1@parsedef\meaning#1\relax
111 \def\bb1@tempc{#2}%
112 \edef\bb1@tempc{\expandafter\strip@prefix\meaning\bb1@tempc}%
113 \def\bb1@tempd{#3}%
114 \edef\bb1@tempd{\expandafter\strip@prefix\meaning\bb1@tempd}%
115 \bb1@xin@\bb1@tempc{\bb1@tempe}% If not in macro, do nothing
116 \ifin@
117 \bb1@exp{\bb1@replace\bb1@tempe{\bb1@tempc}{\bb1@tempd}}%
118 \def\bb1@tempc{% Expanded an executed below as 'uplevel'
119 \\\makeatletter % "internal" macros with @ are assumed
120 \\\scantokens{%
121 \bb1@tempa\\\@namedef{\bb1@stripslash#1}\bb1@tempb{\bb1@tempe}}%
122 \catcode64=\the\catcode64\relax}% Restore @
123 \else
124 \let\bb1@tempc\@empty % Not \relax
125 \fi
126 \bb1@exp{% For the 'uplevel' assignments
127 \endgroup
128 \bb1@tempc}} % empty or expand to set #1 with changes
129 \fi

```

Two further tools. `\bb1@samestring` first expand its arguments and then compare their expansion (sanitized, so that the catcodes do not matter). `\bb1@engine` takes the following values: 0 is pdf $\TeX$ , 1 is luatex, and 2 is xetex. You may use the latter in your language style if you want.

```

130 \def\bb1@ifsamestring#1#2{%
131 \begingroup
132 \protected@edef\bb1@tempb{#1}%
133 \edef\bb1@tempb{\expandafter\strip@prefix\meaning\bb1@tempb}%
134 \protected@edef\bb1@tempc{#2}%
135 \edef\bb1@tempc{\expandafter\strip@prefix\meaning\bb1@tempc}%
136 \ifx\bb1@tempb\bb1@tempc
137 \aftergroup\@firstoftwo
138 \else
139 \aftergroup\@secondoftwo
140 \fi
141 \endgroup}
142 \chardef\bb1@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>> \equiv
154 \ifx\ProvidesFile\@undefined
155 \def\ProvidesFile#1[#2 #3 #4]{%
156 \wlog{File: #1 #4 #3 <#2>}%
157 \let\ProvidesFile\@undefined}
158 \fi
159 <</Make sure ProvidesFile is defined>>

```



## 7.1 Multiple languages

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

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

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

`\addlanguage` This macro was introduced for  $\TeX$  < 2. Preserved for compatibility.

```
165 <<*Define core switching macros>> ≡
166 <<*Define core switching macros>> ≡
167 \countdef\last@language=19 % TODO. why? remove?
168 \def\addlanguage{\csname newlanguage\endcsname}
169 <</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.

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

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

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

The first two options are for debugging.

```
170 <*package>
171 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
172 \ProvidesPackage{babel}[\<date>] [\<version>] The Babel package]
173 \@ifpackagewith{babel}{debug}
174   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}}%
175   \let\bbl@debug\@firstofone}
176   {\providecommand\bbl@trace[1]{}%
177   \let\bbl@debug\@gobble}
178 <<Basic macros>>
179 % Temporarily repeat here the code for errors
180 \def\bbl@error#1#2{%
181   \begingroup
182     \def\{\MessageBreak}%
183     \PackageError{babel}{#1}{#2}%
```

```

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

```

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

```

240 \ifx\bbl@languages\undefined\else
241   \begingroup
242     \catcode`\^^I=12
243     \@ifpackagewith{babel}{showlanguages}{%
244       \begingroup
245         \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
246         \wlog{<*languages>}%
247         \bbl@languages
248         \wlog{</languages>}%
249       \endgroup}{%
250     \endgroup
251     \def\bbl@elt#1#2#3#4{%
252       \ifnum#2=\z@
253         \gdef\bbl@nulllanguage{#1}%
254         \def\bbl@elt##1##2##3##4{%
255           \fi}%
256       \bbl@languages
257     \fi%

```

### 7.3 base

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

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

```

258 \bbl@trace{Defining option 'base'}
259 \@ifpackagewith{babel}{base}{%
260   \let\bbl@onlyswitch\@empty
261   \let\bbl@provide@locale\relax
262   \input babel.def
263   \let\bbl@onlyswitch\@undefined
264   \ifx\directlua\undefined
265     \DeclareOption*{\bbl@patterns{\CurrentOption}}%
266   \else
267     \input luababel.def
268     \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
269   \fi
270   \DeclareOption{base}{}%
271   \DeclareOption{showlanguages}{}%
272   \ProcessOptions
273   \global\expandafter\let\csname opt@babel.sty\endcsname\relax
274   \global\expandafter\let\csname ver@babel.sty\endcsname\relax
275   \global\let\@ifl@ter@\@ifl@ter
276   \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
277   \endinput}{}%
278 % \end{macrocode}
279 %
280 % \subsection{\texttt{key=value} options and other general option}
281 %
282 %   The following macros extract language modifiers, and only real
283 %   package options are kept in the option list. Modifiers are saved
284 %   and assigned to |\BabelModifiers| at |\bbl@load@language|; when
285 %   no modifiers have been given, the former is |\relax|. How
286 %   modifiers are handled are left to language styles; they can use
287 %   |\in@|, loop them with |\@for| or load |keyval|, for example.
288 %

```

```

289% \begin{macrocode}
290 \bbl@trace{key=value and another general options}
291 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
292 \def\bbl@tempb#1.#2{%
293   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
294 \def\bbl@tempd#1.#2@nnil{%
295   \ifx\@empty#2%
296     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
297   \else
298     \in@{=}{#1}\ifin@
299     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
300   \else
301     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
302     \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
303   \fi
304 \fi}
305 \let\bbl@tempc\@empty
306 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
307 \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.

```

308 \DeclareOption{KeepShorthandsActive}{}
309 \DeclareOption{activeacute}{}
310 \DeclareOption{activegrave}{}
311 \DeclareOption{debug}{}
312 \DeclareOption{noconfigs}{}
313 \DeclareOption{showlanguages}{}
314 \DeclareOption{silent}{}
315 \DeclareOption{mono}{}
316 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
317 % Don't use. Experimental. TODO.
318 \newif\ifbbl@single
319 \DeclareOption{selectors=off}{\bbl@singletrue}
320 <<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.

```

321 \let\bbl@opt@shorthands\@nnil
322 \let\bbl@opt@config\@nnil
323 \let\bbl@opt@main\@nnil
324 \let\bbl@opt@headfoot\@nnil
325 \let\bbl@opt@layout\@nnil

```

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

```

326 \def\bbl@tempa#1=#2\bbl@tempa{%
327   \bbl@csarg\ifx{opt@#1}\@nnil
328     \bbl@csarg\edef{opt@#1}{#2}%
329   \else
330     \bbl@error
331     {Bad option `#1=#2'. Either you have misspelled the\\%
332     key or there is a previous setting of `#1'. Valid\\%
333     keys are, among others, `shorthands', `main', `bidi',\\%

```

```

334     'strings', 'config', 'headfoot', 'safe', 'math'.}%
335     {See the manual for further details.}
336   \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.

```

337 \let\bbl@language@opts\@empty
338 \DeclareOption*{%
339   \bbl@xin@{\string=}{\CurrentOption}%
340   \ifin@
341     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
342   \else
343     \bbl@add@list\bbl@language@opts{\CurrentOption}%
344   \fi}

```

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

```

345 \ProcessOptions*

```

## 7.4 Conditional loading of shorthands

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

```

346 \bbl@trace{Conditional loading of shorthands}
347 \def\bbl@sh@string#1{%
348   \ifx#1\@empty\else
349     \ifx#1t\string~%
350     \else\ifx#1c\string,%
351     \else\string#1%
352   \fi\fi
353   \expandafter\bbl@sh@string
354 \fi}
355 \ifx\bbl@opt@shorthands\@nnil
356   \def\bbl@ifshorthand#1#2#3{#2}%
357 \else\ifx\bbl@opt@shorthands\@empty
358   \def\bbl@ifshorthand#1#2#3{#3}%
359 \else

```

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

```

360 \def\bbl@ifshorthand#1{%
361   \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
362   \ifin@
363     \expandafter\@firstoftwo
364   \else
365     \expandafter\@secondoftwo
366   \fi}

```

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

```

367 \edef\bbl@opt@shorthands{%
368   \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.

```
369 \bbl@ifshorthand{'}%
370   {\PassOptionsToPackage{activeacute}{babel}}{}
371 \bbl@ifshorthand{`}%
372   {\PassOptionsToPackage{activegrave}{babel}}{}
373 \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.

```
374 \ifx\bbl@opt@headfoot\@nnil\else
375   \g@addto@macro\@resetactivechars{%
376     \set@typeset@protect
377     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
378     \let\protect\noexpand}
379 \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.

```
380 \ifx\bbl@opt@safe\@undefined
381   \def\bbl@opt@safe{BR}
382 \fi
383 \ifx\bbl@opt@main\@nnil\else
384   \def\bbl@language@opts{%
385     \ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
386     \bbl@opt@main}
387 \fi
```

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

```
388 \bbl@trace{Defining IfBabelLayout}
389 \ifx\bbl@opt@layout\@nnil
390   \newcommand\IfBabelLayout[3]{#3}%
391 \else
392   \newcommand\IfBabelLayout[1]{%
393     \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
394     \ifin@
395       \expandafter\@firstoftwo
396     \else
397       \expandafter\@secondoftwo
398     \fi}
399 \fi
```

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

```
400 \input babel.def
```

## 7.5 Cross referencing macros

The  $\LaTeX$  book states:

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

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

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

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

```
401 <<*More package options>> ≡
402 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
403 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
404 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
405 <</More package options>>
```

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

```
406 \bbl@trace{Cross referencing macros}
407 \ifx\bbl@opt@safe\@empty\else
408   \def\@newl@bel#1#2#3{%
409     {\@safe@activestrue
410       \bbl@ifunset{#1@#2}%
411         \relax
412         {\gdef\@multiplelabels{%
413           \@latex@warning@no@line{There were multiply-defined labels}}%
414           \@latex@warning@no@line{Label `#2' multiply defined}}%
415       \global\@namedef{#1@#2}{#3}}}
```

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

```
416 \CheckCommand*\@testdef[3]{%
417   \def\reserved@a{#3}%
418   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
419   \else
420     \@tempwattrue
421   \fi}
```

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

```
422 \def\@testdef#1#2#3{% TODO. With @samestring?
423   \@safe@activestrue
424   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
425   \def\bbl@tempb{#3}%
426   \@safe@activesfalse
427   \ifx\bbl@tempa\relax
428   \else
429     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
430   \fi
431   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
432   \ifx\bbl@tempa\bbl@tempb
433   \else
434     \@tempwattrue
435   \fi}
436 \fi
```

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

```

437 \bbl@xin@{R}\bbl@opt@safe
438 \ifin@
439 \bbl@redefineroobust\ref#1{%
440 \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
441 \bbl@redefineroobust\pageref#1{%
442 \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
443 \else
444 \let\org@ref\ref
445 \let\org@pageref\pageref
446 \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.

```

447 \bbl@xin@{B}\bbl@opt@safe
448 \ifin@
449 \bbl@redefine\@citex[#1]#2{%
450 \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
451 \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.

```

452 \AtBeginDocument{%
453 \@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.)

```

454 \def\@citex[#1][#2]#3{%
455 \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
456 \org@@citex[#1][#2]{\@tempa}}%
457 }{}

```

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

```

458 \AtBeginDocument{%
459 \@ifpackageloaded{cite}{%
460 \def\@citex[#1]#2{%
461 \@safe@activetrue\org@@citex[#1]{#2}\@safe@activesfalse}%
462 }{}

```

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

```

463 \bbl@redefine\nocite#1{%
464 \@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.

```
465 \bbl@redefine\bibcite{%
466   \bbl@cite@choice
467   \bibcite}
```

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

```
468 \def\bbl@bibcite#1#2{%
469   \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.

```
470 \def\bbl@cite@choice{%
471   \global\let\bibcite\bbl@bibcite
472   \ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
473   \ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
474   \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.

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

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

```
476 \bbl@redefine\@bibitem#1{%
477   \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
478 \else
479   \let\org@nocite\nocite
480   \let\org@@citex\@citex
481   \let\org@bibcite\bibcite
482   \let\org@@bibitem\@bibitem
483 \fi
```

## 7.6 Marks

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

```
484 \bbl@trace{Marks}
485 \IfBabelLayout{sectioning}
486   {\ifx\bbl@opt@headfoot\@nnil
487     \g@addto@macro\@resetactivechars{%
488       \set@typeset@protect
489       \expandafter\select@language@x\expandafter{\bbl@main@language}%
490       \let\protect\noexpand
491       \edef\thepage{% TODO. Only with bidi. See also above
492         \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}}}%
493   \fi}
494 {\ifbbl@single\else
495   \bbl@ifunset{markright} \bbl@redefine\bbl@redefineroobust
496   \markright#1{%
497     \bbl@ifblank{#1}%
```

```

498      {\org@markright{}}}%
499      {\toks@{#1}}%
500      \bbl@exp{%
501        \\org@markright{\\protect\\foreignlanguage{\\language\\name}}%
502        {\\protect\\bbl@restore@actives\\the\\toks@}}}%

```

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

```

503      \ifx\@mkboth\markboth
504        \def\bbl@tempc{\let\@mkboth\markboth}
505      \else
506        \def\bbl@tempc{}
507      \fi
508      \bbl@ifunset{markboth}{\bbl@redefine\bbl@redefineroobust
509      \markboth#1#2}%
510      \protected@edef\bbl@tempb##1{%
511        \protect\foreignlanguage
512        {\language\\name}{\protect\bbl@restore@actives##1}}%
513      \bbl@ifblank{#1}%
514        {\toks@{}}%
515        {\toks@\expandafter{\bbl@tempb{#1}}}%
516      \bbl@ifblank{#2}%
517        {\@temptokena{}}%
518        {\@temptokena\expandafter{\bbl@tempb{#2}}}%
519      \bbl@exp{\\org@markboth{\the\\toks@}{\the\\@temptokena}}
520      \bbl@tempc
521      \fi} % end ifbbl@single, end \IfBabelLayout

```

## 7.7 Preventing clashes with other packages

### 7.7.1 `ifthen`

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

```

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

```

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

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

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

```

522 \bbl@trace{Preventing clashes with other packages}
523 \bbl@xin@{R}\bbl@opt@safe
524 \ifin@
525   \AtBeginDocument{%
526     \@ifpackageloaded{ifthen}{%

```

```

527 \bbl@redefine@long\ifthenelse#1#2#3{%
528 \let\bbl@temp@pref\pageref
529 \let\pageref\org@pageref
530 \let\bbl@temp@ref\ref
531 \let\ref\org@ref
532 \@safe@activestrue
533 \org@ifthenelse{#1}%
534 {\let\pageref\bbl@temp@pref
535 \let\ref\bbl@temp@ref
536 \@safe@activesfalse
537 #2}%
538 {\let\pageref\bbl@temp@pref
539 \let\ref\bbl@temp@ref
540 \@safe@activesfalse
541 #3}%
542 }%
543 }{}%
544 }

```

### 7.7.2 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`.

```

545 \AtBeginDocument{%
546 \ifpackageloaded{varioref}{%
547 \bbl@redefine\@vpageref#1[#2]#3{%
548 \@safe@activestrue
549 \org@@@vpageref{#1}[#2]{#3}%
550 \@safe@activesfalse}%
551 \bbl@redefine\vrefpagemum#1#2{%
552 \@safe@activestrue
553 \org@vrefpagemum{#1}{#2}%
554 \@safe@activesfalse}%

```

The package `varioref` defines `\Ref` to be a robust command which uppercases the first character of the reference text. In order to be able to do that it needs to access the expandable form of `\ref`. So we employ a little trick here. We redefine the (internal) command `\Ref` to call `\org@ref` instead of `\ref`. The disadvantage of this solution is that whenever the definition of `\Ref` changes, this definition needs to be updated as well.

```

555 \expandafter\def\csname Ref \endcsname#1{%
556 \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
557 }{}%
558 }
559 \fi

```

### 7.7.3 hhlne

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

```

560 \AtEndOfPackage{%
561 \AtBeginDocument{%
562 \ifpackageloaded{hhlne}%
563 {\expandafter\ifx\csname normal@char\string\endcsname\relax

```

```

564     \else
565     \makeatletter
566     \def\@currname{hhline}\input{hhline.sty}\makeatother
567     \fi}%
568     {}}}

```

#### 7.7.4 hyperref

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

```

569 \AtBeginDocument{%
570   \ifx\pdfstringdefDisableCommands\undefined\else
571     \pdfstringdefDisableCommands{\languageshorthands{system}}%
572   \fi}

```

#### 7.7.5 fancyhdr

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

```

573 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
574   \lowercase{\foreignlanguage{#1}}}

```

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

```

575 \def\substitutefontfamily#1#2#3{%
576   \lowercase{\immediate\openout15=#1#2.fd\relax}%
577   \immediate\write15{%
578     \string\ProvidesFile{#1#2.fd}%
579     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
580     \space generated font description file]^{}
581     \string\DeclareFontFamily{#1}{#2}{}^{}
582     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^{}
583     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^{}
584     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^{}
585     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^{}
586     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^{}
587     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^{}
588     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^{}
589     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^{}
590   }%
591   \closeout15
592 }
593 \@onlypreamble\substitutefontfamily

```

## 7.8 Encoding and fonts

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

```

594 \bbl@trace{Encoding and fonts}
595 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,PU,PD1}
596 \newcommand\BabelNonText{TS1,T3,TS3}
597 \let\org@TeX\TeX
598 \let\org@LaTeX\LaTeX
599 \let\ensureascii\@firstofone
600 \AtBeginDocument{%
601   \in@false
602   \bbl@foreach\BabelNonASCII{% is there a text non-ascii enc?
603     \ifin@else
604       \lowercase{\bbl@xin@{,#1enc.def,}{,\@filelist,}}%
605     \fi}%
606   \ifin@ % if a text non-ascii has been loaded
607     \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
608     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
609     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
610     \def\bbl@tempb#1\@@{\uppercase{\bbl@tempc#1}ENC.DEF\empty\@@}%
611     \def\bbl@tempc#1ENC.DEF#2\@@{%
612       \ifx\empty#2\else
613         \bbl@ifunset{T#1}%
614         {}%
615         {\bbl@xin@{,#1,}{,\BabelNonASCII,\BabelNonText,}}%
616       \ifin@
617         \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
618         \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
619       \else
620         \def\ensureascii##1{{\fontencoding{#1}\selectfont##1}}%
621       \fi}%
622     \fi}%
623   \bbl@foreach\@filelist{\bbl@tempb#1\@@}% TODO - \@@ de mas??
624   \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
625   \ifin@else
626     \edef\ensureascii#1{%
627       \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}%
628   \fi
629 \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.

```

630 \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.

```

631 \AtBeginDocument{%
632   \ifpackageloaded{fontspec}%
633     {\xdef\latinencoding{%
634       \ifx\UTFencname\@undefined

```

```

635      EU\ifcase\bbl@engine\or2\or1\fi
636      \else
637      \UTFencname
638      \fi}}%
639  {\gdef\latinencoding{OT1}%
640   \ifx\cf@encoding\bbl@t@one
641   \xdef\latinencoding{\bbl@t@one}%
642   \else
643   \ifx\@fontenc@load@list\@undefined
644   \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}}%
645   \else
646   \def\@elt#1{,#1,}%
647   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
648   \let\@elt\relax
649   \bbl@xin@{,T1,}\bbl@tempa
650   \ifin@
651   \xdef\latinencoding{\bbl@t@one}%
652   \fi
653   \fi
654   \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.

```

655 \DeclareRobustCommand{\latintext}{%
656   \fontencoding{\latinencoding}\selectfont
657   \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.

```

658 \ifx\@undefined\DeclareTextFontCommand
659   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
660 \else
661   \DeclareTextFontCommand{\textlatin}{\latintext}
662 \fi

```

## 7.9 Basic bidi support

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

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

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

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

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

As a first step, add a handler for `bidi` and `digits` (and potentially other processes) just before `luaotfload` is applied, which is loaded by default by `LATEX`. Just in case, consider the possibility it has not been loaded.

```

663 \ifodd\bbl@engine
664   \def\bbl@activate@preotf{%
665     \let\bbl@activate@preotf\relax % only once
666     \directlua{
667       Babel = Babel or {}
668       %
669       function Babel.pre_otfload_v(head)
670         if Babel.numbers and Babel.digits_mapped then
671           head = Babel.numbers(head)
672         end
673         if Babel.bidi_enabled then
674           head = Babel.bidi(head, false, dir)
675         end
676         return head
677       end
678       %
679       function Babel.pre_otfload_h(head, gc, sz, pt, dir)
680         if Babel.numbers and Babel.digits_mapped then
681           head = Babel.numbers(head)
682         end
683         if Babel.bidi_enabled then
684           head = Babel.bidi(head, false, dir)
685         end
686         return head
687       end
688       %
689       luatexbase.add_to_callback('pre_linebreak_filter',
690         Babel.pre_otfload_v,
691         'Babel.pre_otfload_v',
692         luatexbase.priority_in_callback('pre_linebreak_filter',
693           'luaotfload.node_processor') or nil)
694       %
695       luatexbase.add_to_callback('hpack_filter',
696         Babel.pre_otfload_h,
697         'Babel.pre_otfload_h',
698         luatexbase.priority_in_callback('hpack_filter',
699           'luaotfload.node_processor') or nil)
700     }}
701 \fi

```

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

```

702 \bbl@trace{Loading basic (internal) bidi support}
703 \ifodd\bbl@engine
704   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
705     \let\bbl@beforeforeign\leavevmode
706     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
707     \RequirePackage{luatexbase}
708     \bbl@activate@preotf
709     \directlua{

```

```

710     require('babel-data-bidi.lua')
711     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
712         require('babel-bidi-basic.lua')
713     \or
714         require('babel-bidi-basic-r.lua')
715     \fi}
716 % TODO - to locale_props, not as separate attribute
717 \newattribute\bbl@attr@dir
718 % TODO. I don't like it, hackish:
719 \bbl@exp{\output{\bodydir\pagedir\the\output}}
720 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
721 \fi\fi
722 \else
723 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
724     \bbl@error
725     {The bidi method 'basic' is available only in\%
726     luatex. I'll continue with 'bidi=default', so\%
727     expect wrong results}%
728     {See the manual for further details.}%
729     \let\bbl@beforeforeign\leavevmode
730     \AtEndOfPackage{%
731         \EnableBabelHook{babel-bidi}%
732         \bbl@xebidipar}
733 \fi\fi
734 \def\bbl@loadxebidi#1{%
735     \ifx\RTLfootnotetext\undefined
736         \AtEndOfPackage{%
737             \EnableBabelHook{babel-bidi}%
738             \ifx\fontspec\undefined
739                 \usepackage{fontspec}% bidi needs fontspec
740             \fi
741             \usepackage#1{bidi}}%
742     \fi}
743 \ifnum\bbl@bidimode>200
744     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
745         \bbl@tentative{bidi=bidi}
746         \bbl@loadxebidi{}
747     \or
748         \bbl@tentative{bidi=bidi-r}
749         \bbl@loadxebidi{[rldocument]}
750     \or
751         \bbl@tentative{bidi=bidi-l}
752         \bbl@loadxebidi{}
753     \fi
754 \fi
755 \fi
756 \ifnum\bbl@bidimode=\@ne
757     \let\bbl@beforeforeign\leavevmode
758     \ifodd\bbl@engine
759         \newattribute\bbl@attr@dir
760         \bbl@exp{\output{\bodydir\pagedir\the\output}}%
761     \fi
762     \AtEndOfPackage{%
763         \EnableBabelHook{babel-bidi}%
764         \ifodd\bbl@engine\else
765             \bbl@xebidipar
766         \fi}
767 \fi

```



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

```

768 \bbl@trace{Macros to switch the text direction}
769 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
770 \def\bbl@rscripts{% TODO. Base on codes ??
771   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
772   Old Hungarian,Old Hungarian,Lybian,Mandaean,Manichaeen,%
773   Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
774   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
775   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
776   Old South Arabian,}%
777 \def\bbl@provide@dirs#1{%
778   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
779   \ifin@
780     \global\bbl@csarg\chardef{wdir@#1}\@ne
781     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
782     \ifin@
783       \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
784     \fi
785   \else
786     \global\bbl@csarg\chardef{wdir@#1}\z@
787   \fi
788   \ifodd\bbl@engine
789     \bbl@csarg\ifcase{wdir@#1}%
790       \directlua{ Babel.locale_props[\the\localeid].texdir = 'l' }%
791     \or
792       \directlua{ Babel.locale_props[\the\localeid].texdir = 'r' }%
793     \or
794       \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
795     \fi
796   \fi}
797 \def\bbl@switchdir{%
798   \bbl@ifunset{bbl@sys@\languagename}{\bbl@provide@sys{\languagename}}{}%
799   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
800   \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}%
801 \def\bbl@setdirs#1{% TODO - math
802   \ifcase\bbl@select@type % TODO - strictly, not the right test
803     \bbl@bodydir{#1}%
804     \bbl@pardir{#1}%
805   \fi
806   \bbl@texdir{#1}}
807 % TODO. Only if \bbl@bidimode > 0?:
808 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
809 \DisableBabelHook{babel-bidi}

```

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

```

810 \ifodd\bbl@engine % luatex=1
811   \chardef\bbl@thetextdir\z@
812   \chardef\bbl@thepardir\z@
813   \def\bbl@getluadir#1{%
814     \directlua{
815       if tex.#1dir == 'TLT' then
816         tex.sprint('0')
817       elseif tex.#1dir == 'TRT' then
818         tex.sprint('1')
819       end}}
820   \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\texdir.. 3=0 lr/1 rl
821     \ifcase#3relax

```

```

822     \ifcase\bb1@getluadir{#1}\relax\else
823     #2 TLT\relax
824     \fi
825   \else
826     \ifcase\bb1@getluadir{#1}\relax
827     #2 TRT\relax
828     \fi
829   \fi}
830 \def\bb1@textdir#1{%
831   \bb1@setluadir{text}\textdir{#1}%
832   \chardef\bb1@thetextdir#1\relax
833   \setattribute\bb1@attr@dir{\numexpr\bb1@thepardir*3+#1}}
834 \def\bb1@pardir#1{%
835   \bb1@setluadir{par}\pardir{#1}%
836   \chardef\bb1@thepardir#1\relax}
837 \def\bb1@bodydir{\bb1@setluadir{body}\bodydir}
838 \def\bb1@pagedir{\bb1@setluadir{page}\pagedir}
839 \def\bb1@dirparastext{\pardir\the\textdir\relax}%   %%%
840 % Sadly, we have to deal with boxes in math with basic.
841 % Activated every math with the package option bidi=:
842 \def\bb1@mathboxdir{%
843   \ifcase\bb1@thetextdir\relax
844     \everyhbox{\textdir TLT\relax}%
845   \else
846     \everyhbox{\textdir TRT\relax}%
847   \fi}
848 \frozen@everymath\expandafter{%
849   \expandafter\bb1@mathboxdir\the\frozen@everymath}
850 \frozen@everydisplay\expandafter{%
851   \expandafter\bb1@mathboxdir\the\frozen@everydisplay}
852 \else % pdftex=0, xetex=2
853   \newcount\bb1@dirlevel
854   \chardef\bb1@thetextdir\z@
855   \chardef\bb1@thepardir\z@
856   \def\bb1@textdir#1{%
857     \ifcase#1\relax
858       \chardef\bb1@thetextdir\z@
859       \bb1@textdir@i\beginL\endL
860     \else
861       \chardef\bb1@thetextdir\@ne
862       \bb1@textdir@i\beginR\endR
863     \fi}
864   \def\bb1@textdir@i#1#2{%
865     \ifhmode
866       \ifnum\currentgrouplevel>\z@
867         \ifnum\currentgrouplevel=\bb1@dirlevel
868           \bb1@error{Multiple bidi settings inside a group}%
869           {I'll insert a new group, but expect wrong results.}%
870           \bgroup\aftergroup#2\aftergroup\egroup
871         \else
872           \ifcase\currentgrouptype\or % 0 bottom
873             \aftergroup#2% 1 simple {}
874           \or
875             \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
876           \or
877             \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
878           \or\or\or % vbox vtop align
879           \or
880             \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign

```

```

881      \or\or\or\or\or\or % output math disc insert vcent mathchoice
882      \or
883      \aftergroup#2% 14 \begingroup
884      \else
885      \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
886      \fi
887      \fi
888      \bbl@dirlevel\currentgrouplevel
889      \fi
890      #1%
891      \fi}
892 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
893 \let\bbl@bodydir\@gobble
894 \let\bbl@pagedir\@gobble
895 \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).

```

896 \def\bbl@xebidipar{%
897   \let\bbl@xebidipar\relax
898   \TeXeTstate\@ne
899   \def\bbl@xeverypar{%
900     \ifcase\bbl@thepardir
901       \ifcase\bbl@thetextdir\else\beginR\fi
902     \else
903       {\setbox\z@\lastbox\beginR\box\z@}%
904     \fi}%
905   \let\bbl@severypar\everypar
906   \newtoks\everypar
907   \everypar=\bbl@severypar
908   \bbl@severypar{\bbl@xeverypar\the\everypar}}
909 \ifnum\bbl@bidimode>200
910   \let\bbl@textdir\i\@gobbletwo
911   \let\bbl@xebidipar\@empty
912   \AddBabelHook{bidi}{foreign}{%
913     \def\bbl@tempa{\def\BabelText####1}%
914     \ifcase\bbl@thetextdir
915       \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
916     \else
917       \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
918     \fi}
919   \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
920 \fi
921 \fi

```

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

```

922 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
923 \AtBeginDocument{%
924   \ifx\pdfstringdefDisableCommands\@undefined\else
925     \ifx\pdfstringdefDisableCommands\relax\else
926       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
927     \fi
928   \fi}

```

## 7.10 Local Language Configuration

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

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

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

```

929 \bbl@trace{Local Language Configuration}
930 \ifx\loadlocalcfg\undefined
931   \@ifpackagewith{babel}{noconfigs}%
932   {\let\loadlocalcfg@gobble}%
933   {\def\loadlocalcfg#1{%
934     \InputIfFileExists{#1.cfg}%
935     {\typeout{*****^J%
936               * Local config file #1.cfg used^^J%
937               *}}%
938     \@empty}}
939 \fi

```

Just to be compatible with L<sup>A</sup>T<sub>E</sub>X 2.09 we add a few more lines of code. TODO. Necessary? Correct place? Used by some ldf file?

```

940 \ifx\@unexpandable@protect\undefined
941   \def\@unexpandable@protect{\noexpand\protect\noexpand}
942   \long\def\protected@write#1#2#3{%
943     \begingroup
944       \let\thepage\relax
945       #2%
946       \let\protect\@unexpandable@protect
947       \edef\reserved@a{\write#1{#3}}%
948       \reserved@a
949     \endgroup
950     \if@nobreak\ifvmode\nobreak\fi\fi}
951 \fi
952 %
953 % \subsection{Language options}
954 %
955 % Languages are loaded when processing the corresponding option
956 % \textit{except} if a |main| language has been set. In such a
957 % case, it is not loaded until all options has been processed.
958 % The following macro inputs the ldf file and does some additional
959 % checks (\input works, too, but possible errors are not caught).
960 %
961 % \begin{macrocode}
962 \bbl@trace{Language options}
963 \let\bbl@afterlang\relax
964 \let\BabelModifiers\relax
965 \let\bbl@loaded\@empty
966 \def\bbl@load@language#1{%
967   \InputIfFileExists{#1.ldf}%
968   {\edef\bbl@loaded{\CurrentOption
969     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
970     \expandafter\let\expandafter\bbl@afterlang
971     \csname\CurrentOption.ldf-h@@k\endcsname
972     \expandafter\let\expandafter\BabelModifiers
973     \csname bbl@mod@\CurrentOption\endcsname}%
974   {\bbl@error{%
975     Unknown option '\CurrentOption'. Either you misspelled it\\%
976     or the language definition file \CurrentOption.ldf was not found}}%
977     Valid options are: shorthands=, KeepShorthandsActive,\\%
978     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
979     headfoot=, strings=, config=, hyphenmap=, or a language name.}}}

```

Now, we set a few language options whose names are different from ldf files. These declarations are preserved for backwards compatibility, but they must be eventually removed. Use proxy files instead.

```

980 \def\bbl@try@load@lang#1#2#3{%
981   \IfFileExists{\CurrentOption.lda}%
982     {\bbl@load@language{\CurrentOption}}%
983     {#1\bbl@load@language{#2}#3}}
984 \DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}{}}
985 \DeclareOption{hebrew}{%
986   \input{rlbabel.def}%
987   \bbl@load@language{hebrew}}
988 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
989 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
990 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
991 \DeclareOption{polutonikogreek}{%
992   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
993 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
994 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
995 \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.

```

996 \ifx\bbl@opt@config\@nnil
997   \@ifpackagewith{babel}{noconfigs}{}%
998   {\InputIfFileExists{bblopts.cfg}%
999     {\typeout{*****^J%
1000              * Local config file bblopts.cfg used^^J%
1001              *}}%
1002     {}}%
1003 \else
1004   \InputIfFileExists{\bbl@opt@config.cfg}%
1005   {\typeout{*****^J%
1006            * Local config file \bbl@opt@config.cfg used^^J%
1007            *}}%
1008   {\bbl@error{%
1009     Local config file '\bbl@opt@config.cfg' not found}{%
1010     Perhaps you misspelled it.}}%
1011 \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.

```

1012 \bbl@for\bbl@tempa\bbl@language@opts{%
1013   \bbl@ifunset{ds@\bbl@tempa}%
1014   {\edef\bbl@tempb{%
1015     \noexpand\DeclareOption
1016     {\bbl@tempa}%
1017     {\noexpand\bbl@load@language{\bbl@tempa}}}%
1018     \bbl@tempb}%
1019   \@empty}
```

Now, we make sure an option is explicitly declared for any language set as global option, by checking if an lda 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.

```

1020 \bbl@foreach\@classoptionslist{%
1021   \bbl@ifunset{ds@#1}%
1022     {\IfFileExists{#1.ldf}%
1023       {\DeclareOption{#1}{\bbl@load@language{#1}}}%
1024       {}}%
1025   {}}

```

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

```

1026 \ifx\bbl@opt@main\@nnil\else
1027   \expandafter
1028   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
1029   \DeclareOption{\bbl@opt@main}{}
1030 \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):

```

1031 \def\AfterBabelLanguage#1{%
1032   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
1033 \DeclareOption*{}
1034 \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.

```

1035 \bbl@trace{Option 'main'}
1036 \ifx\bbl@opt@main\@nnil
1037   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
1038   \let\bbl@tempc\@empty
1039   \bbl@for\bbl@tempb\bbl@tempa{%
1040     \bbl@xin@{\bbl@tempb},{\bbl@loaded},}%
1041     \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
1042   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
1043   \expandafter\bbl@tempa\bbl@loaded,\@nnil
1044   \ifx\bbl@tempb\bbl@tempc\else
1045     \bbl@warning{%
1046       Last declared language option is '\bbl@tempc',\%
1047       but the last processed one was '\bbl@tempb'.\%
1048       The main language cannot be set as both a global\%
1049       and a package option. Use 'main=\bbl@tempc' as\%
1050       option. Reported}%
1051   \fi
1052 \else
1053   \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
1054   \ExecuteOptions{\bbl@opt@main}
1055   \DeclareOption*{}
1056   \ProcessOptions*
1057 \fi
1058 \def\AfterBabelLanguage{%
1059   \bbl@error
1060     {Too late for \string\AfterBabelLanguage}%
1061     {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.

```

1062 \ifx\bbl@main@language\@undefined
1063   \bbl@info{%
1064     You haven't specified a language. I'll use 'nil'\%
1065     as the main language. Reported}
1066   \bbl@load@language{nil}
1067 \fi
1068 \</package>
1069 \<core>

```

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

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

### 8.1 Tools

```

1070 \ifx\ldf@quit\@undefined\else
1071 \endinput\fi % Same line!
1072 <<Make sure ProvidesFile is defined>>
1073 \ProvidesFile{babel.def}[\<date>] [\<version>] Babel common definitions]

```

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

```

1074 \ifx\AtBeginDocument\@undefined % TODO. change test.
1075   <<Emulate LaTeX>>
1076   \def\language{english}%
1077   \let\bbl@opt@shorthands\@nnil
1078   \def\bbl@ifshorthand#1#2#3{#2}%
1079   \let\bbl@language@opts\@empty
1080   \ifx\babeloptionstrings\@undefined
1081     \let\bbl@opt@strings\@nnil
1082   \else
1083     \let\bbl@opt@strings\babeloptionstrings
1084   \fi
1085   \def\BabelStringsDefault{generic}
1086   \def\bbl@tempa{normal}
1087   \ifx\babeloptionmath\bbl@tempa
1088     \def\bbl@mathnormal{\noexpand\textormath}
1089   \fi
1090   \def\AfterBabelLanguage#1#2{}
1091   \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi

```

```

1092 \let\bbl@afterlang\relax
1093 \def\bbl@opt@safe{BR}
1094 \ifx\uclclist\@undefined\let\uclclist\@empty\fi
1095 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
1096 \expandafter\newif\csname ifbbl@single\endcsname
1097 \chardef\bbl@bidimode\z@
1098 \fi

```

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

```

1099 \ifx\bbl@trace\@undefined
1100 \let\LdfInit\endinput
1101 \def\ProvidesLanguage#1{\endinput}
1102 \endinput\fi % Same line!

```

And continue.

## 9 Multiple languages

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

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

```

1103 <<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.

```

1104 \def\bbl@version{<<version>>}}
1105 \def\bbl@date{<<date>>}}
1106 \def\adddialect#1#2{%
1107   \global\chardef#1#2\relax
1108   \bbl@usehooks{adddialect}{#1}{#2}}%
1109 \begingroup
1110   \count@#1\relax
1111   \def\bbl@elt##1##2##3##4{%
1112     \ifnum\count@=##2\relax
1113       \bbl@info{\string#1 = using hyphenrules for ##1\%
1114         (\string\language\the\count@)}%
1115       \def\bbl@elt####1####2####3####4{%
1116         \fi}%
1117       \bbl@cs{languages}%
1118     \endgroup}

```

`\bbl@iflanguage` executes code only if the language `l@` exists. Otherwise raises and error. The argument of `\bbl@fixname` has to be a macro name, as it may get “fixed” if casing (lc/uc) is wrong. It’s intended to fix a long-standing bug when `\foreignlanguage` and the like appear in a `\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.

```

1119 \def\bbl@fixname#1{%
1120   \begingroup
1121     \def\bbl@tempe{l@}%
1122     \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
1123     \bbl@tempd
1124     {\lowercase\expandafter{\bbl@tempd}}%
1125     {\uppercase\expandafter{\bbl@tempd}}%
1126     \@empty

```



```

1127         {\edef\bbl@tempd{\def\noexpand#1{#1}}}%
1128         \uppercase\expandafter{\bbl@tempd}}}%
1129         {\edef\bbl@tempd{\def\noexpand#1{#1}}}%
1130         \lowercase\expandafter{\bbl@tempd}}}%
1131     \@empty
1132     \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
1133     \bbl@tempd
1134     \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}%
1135     \def\bbl@iflanguage#1{%
1136     \@ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}

```

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

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

```

1137 \def\bbl@bcpcase#1#2#3#4\@#5{%
1138     \ifx\@empty#3%
1139         \uppercase{\def#5{#1#2}}%
1140     \else
1141         \uppercase{\def#5{#1}}%
1142         \lowercase{\edef#5{#5#2#3#4}}%
1143     \fi}
1144 \def\bbl@bcpllookup#1-#2-#3-#4\@{%
1145     \let\bbl@bcp\relax
1146     \lowercase{\def\bbl@tempa{#1}}%
1147     \ifx\@empty#2%
1148         \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1149     \else\ifx\@empty#3%
1150         \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
1151         \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
1152         {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}}%
1153         {}%
1154     \ifx\bbl@bcp\relax
1155         \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1156     \fi
1157     \else
1158         \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
1159         \bbl@bcpcase#3\@empty\@empty\@{\bbl@tempc
1160         \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
1161         {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}}%
1162         {}%
1163     \ifx\bbl@bcp\relax
1164         \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1165         {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}}%
1166         {}%
1167     \fi
1168     \ifx\bbl@bcp\relax
1169         \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1170         {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}}%
1171         {}%
1172     \fi
1173     \ifx\bbl@bcp\relax
1174         \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1175     \fi
1176     \fi\fi}

```

```

1177 \let\bbl@autoload@options\@empty
1178 \let\bbl@initoload\relax
1179 \def\bbl@provide@locale{%
1180   \ifx\babelprovide\undefined
1181     \bbl@error{For a language to be defined on the fly 'base'\\%
1182               is not enough, and the whole package must be\\%
1183               loaded. Either delete the 'base' option or\\%
1184               request the languages explicitly}%
1185     {See the manual for further details.}%
1186   \fi
1187 % TODO. Option to search if loaded, with \LocaleForEach
1188 \let\bbl@auxname\language % Still necessary. TODO
1189 \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
1190   {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
1191 \ifbbl@bcpallowed
1192   \expandafter\ifx\csname date\language\endcsname\relax
1193     \expandafter
1194     \bbl@bcplookup\language-\@empty-\@empty-\@empty\@@
1195     \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
1196       \edef\language{\bbl@bcp@prefix\bbl@bcp}%
1197       \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
1198       \expandafter\ifx\csname date\language\endcsname\relax
1199         \let\bbl@initoload\bbl@bcp
1200         \bbl@exp{\bbl@babelprovide[\bbl@autoload@bcpoptions]{\language}}%
1201         \let\bbl@initoload\relax
1202       \fi
1203       \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
1204     \fi
1205   \fi
1206 \fi
1207 \expandafter\ifx\csname date\language\endcsname\relax
1208   \IfFileExists{babel-\language.tex}%
1209   {\bbl@exp{\bbl@babelprovide[\bbl@autoload@options]{\language}}}%
1210   {}%
1211 \fi}

```

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

```

1212 \def\iflanguage#1{%
1213   \bbl@iflanguage{#1}%
1214   \ifnum\csname l@#1\endcsname=\language
1215     \expandafter\@firstoftwo
1216   \else
1217     \expandafter\@secondoftwo
1218   \fi}}

```

## 9.1 Selecting the language

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

```

1219 \let\bbl@select@type\z@
1220 \edef\selectlanguage{%
1221   \noexpand\protect

```

```
1222 \expandafter\noexpand\csname selectlanguage \endcsname}
```

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

```
1223 \ifx\@undefined\protect\let\protect\relax\fi
```

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

```
1224 \let\xstring\string
```

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

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

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

```
1225 \def\bbl@language@stack{}
```

When using a stack we need a mechanism to push an element on the stack and to retrieve the information afterwards.

`\bbl@push@language` The stack is simply a list of languagenames, separated with a '+' sign; the push function can be simple:

`\bbl@pop@language`

```
1226 \def\bbl@push@language{%
1227   \ifx\language\@undefined\else
1228     \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1229   \fi}
```

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

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

```
1230 \def\bbl@pop@lang#1+#2&#3{%
1231   \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.

```
1232 \let\bbl@ifrestoring\@secondoftwo
1233 \def\bbl@pop@language{%
1234   \expandafter\bbl@pop@lang\bbl@language@stack&\bbl@language@stack
1235   \let\bbl@ifrestoring\@firstoftwo
1236   \expandafter\bbl@set@language\expandafter{\language}%
1237   \let\bbl@ifrestoring\@secondoftwo}
```

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

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

```

1238 \chardef\localeid\z@
1239 \def\bbl@id@last{0} % No real need for a new counter
1240 \def\bbl@id@assign{%
1241   \bbl@ifunset{bbl@id@@\language}%
1242   {\count@bbl@id@last\relax
1243    \advance\count@ne
1244    \bbl@csarg\chardef{id@@\language}\count@
1245    \edef\bbl@id@last{\the\count@}%
1246    \ifcase\bbl@engine\or
1247      \directlua{
1248        Babel = Babel or {}
1249        Babel.locale_props = Babel.locale_props or {}
1250        Babel.locale_props[\bbl@id@last] = {}
1251        Babel.locale_props[\bbl@id@last].name = '\language'
1252      }%
1253    \fi}%
1254  }%
1255  \chardef\localeid\bbl@cl{id@}}

```

The unprotected part of `\selectlanguage`.

```

1256 \expandafter\def\csname selectlanguage \endcsname#1{%
1257   \ifnum\bbl@hymapsel=\@ccclv\let\bbl@hymapsel\tw@fi
1258   \bbl@push@language
1259   \aftergroup\bbl@pop@language
1260   \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.

```

1261 \def\BabelContentsFiles{toc,lof,lot}
1262 \def\bbl@set@language#1{% from selectlanguage, pop@
1263   % The old buggy way. Preserved for compatibility.
1264   \edef\language{%
1265     \ifnum\escapechar=\expandafter`\string#1\@empty
1266     \else\string#1\@empty\fi}%
1267   \ifcat\relax\noexpand#1%
1268     \expandafter\ifx\csname date\language\endcsname\relax
1269       \edef\language{#1}%
1270       \let\localename\language
1271     \else
1272       \bbl@info{Using '\string\language' instead of 'language' is%%
1273         deprecated. If what you want is to use a%%
1274         macro containing the actual locale, make%%
1275         sure it does not not match any language.%%
1276         Reported}%

```

```

1277 %                I'll\\%
1278 %                try to fix '\string\localename', but I cannot promise\\%
1279 %                anything. Reported}%
1280 \ifx\scantokens\@undefined
1281 \def\localename{??}%
1282 \else
1283 \scantokens\expandafter{\expandafter
1284 \def\expandafter\localename\expandafter{\language}%
1285 \fi
1286 \fi
1287 \else
1288 \def\localename{#1}% This one has the correct catcodes
1289 \fi
1290 \select@language{\language}%
1291 % write to auxs
1292 \expandafter\ifx\csname date\language\endcsname\relax\else
1293 \if@files
1294 \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
1295 \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
1296 \fi
1297 \bbl@usehooks{write}{}}%
1298 \fi
1299 \fi}
1300 %
1301 \newif\ifbbl@bcpallowed
1302 \bbl@bcpallowedfalse
1303 \def\select@language#1{% from set@, babel@aux
1304 % set hymap
1305 \ifnum\bbl@hymapset=\@cclv\chardef\bbl@hymapset4\relax\fi
1306 % set name
1307 \edef\language{#1}%
1308 \bbl@fixname\language
1309 % TODO. name@map must be here?
1310 \bbl@provide@locale
1311 \bbl@iflanguage\language{%
1312 \expandafter\ifx\csname date\language\endcsname\relax
1313 \bbl@error
1314 {Unknown language '\language'. Either you have\\%
1315 misspelled its name, it has not been installed,\\%
1316 or you requested it in a previous run. Fix its name,\\%
1317 install it or just rerun the file, respectively. In\\%
1318 some cases, you may need to remove the aux file}%
1319 {You may proceed, but expect wrong results}%
1320 \else
1321 % set type
1322 \let\bbl@select@type\z@
1323 \expandafter\bbl@switch\expandafter{\language}%
1324 \fi}}
1325 \def\babel@aux#1#2{%
1326 \select@language{#1}%
1327 \bbl@foreach\BabelContentsFiles{%
1328 \@writefile{##1}{\babel@toc{#1}{#2}}}% % TODO - ok in plain?
1329 \def\babel@toc#1#2{%
1330 \select@language{#1}}

```

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

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

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

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

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

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

```

1331 \newif\ifbbl@usedategroup
1332 \def\bbl@switch#1{% from select@, foreign@
1333   % make sure there is info for the language if so requested
1334   \bbl@ensureinfo{#1}%
1335   % restore
1336   \originalTeX
1337   \expandafter\def\expandafter\originalTeX\expandafter{%
1338     \csname noextras#1\endcsname
1339     \let\originalTeX\empty
1340     \babel@beginsave}%
1341   \bbl@usehooks{afterreset}{}%
1342   \languageshorthands{none}%
1343   % set the locale id
1344   \bbl@id@assign
1345   % switch captions, date
1346   \ifcase\bbl@select@type
1347     \ifhmode
1348       \hskip\z@skip % trick to ignore spaces
1349       \csname captions#1\endcsname\relax
1350       \csname date#1\endcsname\relax
1351       \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
1352     \else
1353       \csname captions#1\endcsname\relax
1354       \csname date#1\endcsname\relax
1355     \fi
1356   \else
1357     \ifhmode
1358       \hskip\z@skip % trick to ignore spaces
1359       \bbl@xin@{,captions,}{, \bbl@select@opts,}%
1360       \ifin@
1361         \csname captions#1\endcsname\relax
1362       \fi
1363       \bbl@xin@{,date,}{, \bbl@select@opts,}%
1364       \ifin@ % if \foreign... within \<lang>date
1365         \csname date#1\endcsname\relax
1366       \fi
1367       \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
1368     \else
1369       \bbl@xin@{,captions,}{, \bbl@select@opts,}%
1370       \ifin@
1371         \csname captions#1\endcsname\relax
1372       \fi
1373       \bbl@xin@{,date,}{, \bbl@select@opts,}%
1374       \ifin@
1375         \csname date#1\endcsname\relax
1376       \fi
1377     \fi
1378   \fi

```

```

1379 % switch extras
1380 \bbl@usehooks{beforeextras}{}%
1381 \csname extras#1\endcsname\relax
1382 \bbl@usehooks{afterextras}{}%
1383 % > babel-ensure
1384 % > babel-sh-<short>
1385 % > babel-bidi
1386 % > babel-fontspec
1387 % hyphenation - case mapping
1388 \ifcase\bbl@opt@hyphenmap\or
1389 \def\BabelLower##1##2{\lcode##1=##2\relax}%
1390 \ifnum\bbl@hymapsel>4\else
1391 \csname\language @bbl@hyphenmap\endcsname
1392 \fi
1393 \chardef\bbl@opt@hyphenmap\z@
1394 \else
1395 \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
1396 \csname\language @bbl@hyphenmap\endcsname
1397 \fi
1398 \fi
1399 \global\let\bbl@hymapsel@cclv
1400 % hyphenation - patterns
1401 \bbl@patterns{#1}%
1402 % hyphenation - mins
1403 \babel@savevariable\lefthyphenmin
1404 \babel@savevariable\righthyphenmin
1405 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1406 \set@hyphenmins\tw@\thr@@\relax
1407 \else
1408 \expandafter\expandafter\expandafter\set@hyphenmins
1409 \csname #1hyphenmins\endcsname\relax
1410 \fi}

```

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

```

1411 \long\def\otherlanguage#1{%
1412 \ifnum\bbl@hymapsel=\cclv\let\bbl@hymapsel\thr@@\fi
1413 \csname selectlanguage \endcsname{#1}%
1414 \ignorespaces}

```

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

```

1415 \long\def\endotherlanguage{%
1416 \global\@ignoretrue\ignorespaces}

```

**otherlanguage\*** The other language 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`.

```

1417 \expandafter\def\csname otherlanguage*\endcsname{%
1418 \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
1419 \def\bbl@otherlanguage@s[#1]#2{%
1420 \ifnum\bbl@hymapsel=\cclv\chardef\bbl@hymapsel4\relax\fi
1421 \def\bbl@select@opts{#1}%
1422 \foreign@language{#2}}

```

At the end of the environment we need to switch off the extra definitions. The grouping mechanism of the environment will take care of resetting the correct hyphenation rules and “extras”.

```
1423 \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`.

```
1424 \providecommand\bbl@beforeforeign{}
1425 \edef\foreignlanguage{%
1426   \noexpand\protect
1427   \expandafter\noexpand\csname foreignlanguage \endcsname}
1428 \expandafter\def\csname foreignlanguage \endcsname{%
1429   \@ifstar\bbl@foreign@s\bbl@foreign@x}
1430 \providecommand\bbl@foreign@x[3][]{%
1431   \begingroup
1432     \def\bbl@select@opts{#1}%
1433     \let\BabelText\@firstofone
1434     \bbl@beforeforeign
1435     \foreign@language{#2}%
1436     \bbl@usehooks{foreign}{}%
1437     \BabelText{#3}% Now in horizontal mode!
1438   \endgroup}
1439 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@@par
1440   \begingroup
1441     {\par}%
1442     \let\BabelText\@firstofone
1443     \foreign@language{#1}%
1444     \bbl@usehooks{foreign*}{}%
1445     \bbl@dirparastext
1446     \BabelText{#2}% Still in vertical mode!
1447     {\par}%
1448   \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`.



```

1449 \def\foreign@language#1{%
1450 % set name
1451 \edef\language{#1}%
1452 \ifbbl@usedategroup
1453 \bbl@add\bbl@select@opts{,date,}%
1454 \bbl@usedategroupfalse
1455 \fi
1456 \bbl@fixname\language
1457 % TODO. name@map here?
1458 \bbl@provide@locale
1459 \bbl@iflanguage\language{
1460 \expandafter\ifx\csname date\language\endcsname\relax
1461 \bbl@warning % TODO - why a warning, not an error?
1462 {Unknown language `#1'. Either you have\\%
1463 misspelled its name, it has not been installed,\\%
1464 or you requested it in a previous run. Fix its name,\\%
1465 install it or just rerun the file, respectively. In\\%
1466 some cases, you may need to remove the aux file.\\%
1467 I'll proceed, but expect wrong results.\\%
1468 Reported}%
1469 \fi
1470 % set type
1471 \let\bbl@select@type\@ne
1472 \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.

```

1473 \let\bbl@hyphlist\empty
1474 \let\bbl@hyphenation\relax
1475 \let\bbl@pttnlist\empty
1476 \let\bbl@patterns\relax
1477 \let\bbl@hymapsel=\cclv
1478 \def\bbl@patterns#1{%
1479 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1480 \csname l@#1\endcsname
1481 \edef\bbl@tempa{#1}%
1482 \else
1483 \csname l@#1:\f@encoding\endcsname
1484 \edef\bbl@tempa{#1:\f@encoding}%
1485 \fi
1486 \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}}%
1487 % > luatex
1488 \@ifundefined{bbl@hyphenation@}{% Can be \relax!
1489 \begingroup
1490 \bbl@xin{, \number\language,}{, \bbl@hyphlist}%
1491 \ifin\else
1492 \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}}%
1493 \hyphenation{%
1494 \bbl@hyphenation@
1495 \@ifundefined{bbl@hyphenation@#1}%
1496 \empty

```

```

1497         {\space\csname bbl@hyphenation@#1\endcsname}}%
1498         \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1499         \fi
1500     \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*`.

```

1501 \def\hyphenrules#1{%
1502     \edef\bbl@tempf{#1}%
1503     \bbl@fixname\bbl@tempf
1504     \bbl@iflanguage\bbl@tempf{%
1505         \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1506         \languageshortands{none}%
1507         \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1508             \set@hyphenmins\tw@\thr@@\relax
1509         \else
1510             \expandafter\expandafter\expandafter\set@hyphenmins
1511             \csname\bbl@tempf hyphenmins\endcsname\relax
1512         \fi}}
1513 \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.

```

1514 \def\providehyphenmins#1#2{%
1515     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1516         \@namedef{#1hyphenmins}{#2}%
1517     \fi}

```

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

```

1518 \def\set@hyphenmins#1#2{%
1519     \lefthyphenmin#1\relax
1520     \righthyphenmin#2\relax}

```

**\ProvidesLanguage** The identification code for each file is something that was introduced in  $\text{\LaTeX 2}_{\epsilon}$ . When the command `\ProvidesFile` does not exist, a dummy definition is provided temporarily. For use in the language definition file the command `\ProvidesLanguage` is defined by `babel`. Depending on the format, ie, on if the former is defined, we use a similar definition or not.

```

1521 \ifx\ProvidesFile\@undefined
1522     \def\ProvidesLanguage#1[#2 #3 #4]{%
1523         \wlog{Language: #1 #4 #3 <#2>}%
1524     }
1525 \else
1526     \def\ProvidesLanguage#1{%
1527         \begingroup
1528         \catcode`\ 10 %
1529         \@makeother\%
1530         \@ifnextchar[%]
1531             {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
1532     \def\@provideslanguage#1[#2]{%
1533         \wlog{Language: #1 #2}%
1534         \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1535     \endgroup}
1536 \fi

```

`\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`.

```
1537 \ifx\originalTeX\undefined\let\originalTeX\@empty\fi
```

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

```
1538 \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’:

```
1539 \providecommand\setlocale{%
1540   \bbl@error
1541   {Not yet available}%
1542   {Find an armchair, sit down and wait}}
1543 \let\uselocale\setlocale
1544 \let\locale\setlocale
1545 \let\selectlocale\setlocale
1546 \let\localename\setlocale
1547 \let\textlocale\setlocale
1548 \let\textlanguage\setlocale
1549 \let\languagegettext\setlocale
```

## 9.2 Errors

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

`\@noopterr` When the package was loaded without options not everything will work as expected. An error message is issued in that case.

When the format knows about `\PackageError` it must be  $\LaTeX 2_{\epsilon}$ , 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.

```
1550 \edef\bbl@nulllanguage{\string\language=0}
1551 \ifx\PackageError\undefined % TODO. Move to Plain
1552   \def\bbl@error#1#2{%
1553     \begingroup
1554       \newlinechar=`^^J
1555       \def\{^^J(babel) }%
1556       \errhelp{#2}\errmessage{\{#1}%
1557     \endgroup}
1558   \def\bbl@warning#1{%
1559     \begingroup
1560       \newlinechar=`^^J
1561       \def\{^^J(babel) }%
1562       \message{\{#1}%
1563     \endgroup}
1564   \let\bbl@infowarn\bbl@warning
1565   \def\bbl@info#1{%
1566     \begingroup
1567       \newlinechar=`^^J
1568     \def\{^^J}%
```

```

1569     \wlog{#1}%
1570 \endgroup}
1571 \fi
1572 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1573 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1574   \global\@namedef{#2}{\textbf{?#1?}}}%
1575   \@nameuse{#2}%
1576   \bbl@warning{%
1577     \@backslashchar#2 not set. Please, define\\%
1578     it in the preamble with something like:\\%
1579     \string\renewcommand\@backslashchar#2{..}\\%
1580     Reported}}
1581 \def\bbl@tentative{\protect\bbl@tentative@i}
1582 \def\bbl@tentative@i#1{%
1583   \bbl@warning{%
1584     Some functions for '#1' are tentative.\\%
1585     They might not work as expected and their behavior\\%
1586     could change in the future.\\%
1587     Reported}}
1588 \def\@nolanerr#1{%
1589   \bbl@error
1590   {You haven't defined the language #1\space yet.\\%
1591     Perhaps you misspelled it or your installation\\%
1592     is not complete}%
1593   {Your command will be ignored, type <return> to proceed}}
1594 \def\@nopatterns#1{%
1595   \bbl@warning
1596   {No hyphenation patterns were preloaded for\\%
1597     the language `#1' into the format.\\%
1598     Please, configure your TeX system to add them and\\%
1599     rebuild the format. Now I will use the patterns\\%
1600     preloaded for \bbl@nulllanguage\space instead}}
1601 \let\bbl@usehooks\@gobbletwo
1602 \ifx\bbl@onlyswitch\@empty\endinput\fi
1603 % Here ended switch.def

Here ended switch.def.

1604 \ifx\directlua\@undefined\else
1605   \ifx\bbl@luapatterns\@undefined
1606     \input luababel.def
1607   \fi
1608 \fi
1609 <<Basic macros>>
1610 \bbl@trace{Compatibility with language.def}
1611 \ifx\bbl@languages\@undefined
1612   \ifx\directlua\@undefined
1613     \openin1 = language.def % TODO. Remove hardcoded number
1614     \ifeof1
1615       \closein1
1616       \message{I couldn't find the file language.def}
1617     \else
1618       \closein1
1619       \begingroup
1620         \def\addlanguage#1#2#3#4#5{%
1621           \expandafter\ifx\csname lang@#1\endcsname\relax\else
1622             \global\expandafter\let\csname l@#1\endcsname
1623             \csname lang@#1\endcsname
1624           \fi}%
1625         \def\uselanguage#1{%

```

```

1626      \input language.def
1627      \endgroup
1628      \fi
1629      \fi
1630      \chardef\l@english\z@
1631      \fi

```

`\addto` It takes two arguments, a *control sequence* and TeX-code to be added to the *control sequence*.  
 If the *control sequence* has not been defined before it is defined now. The control sequence could also expand to `\relax`, in which case a circular definition results. The net result is a stack overflow. Note there is an inconsistency, because the assignment in the last branch is global.

```

1632 \def\addto#1#2{%
1633   \ifx#1\undefined
1634     \def#1{#2}%
1635   \else
1636     \ifx#1\relax
1637       \def#1{#2}%
1638     \else
1639       {\toks@\expandafter{#1#2}%
1640        \xdef#1{\the\toks@}}%
1641     \fi
1642   \fi}

```

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

```

1643 \def\bbl@withactive#1#2{%
1644   \begingroup
1645   \lccode`~=#2\relax
1646   \lowercase{\endgroup#1~}}

```

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

```

1647 \def\bbl@redefine#1{%
1648   \edef\bbl@tempa{\bbl@stripslash#1}%
1649   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1650   \expandafter\def\csname\bbl@tempa\endcsname{
1651     \@onlypreamble\bbl@redefine

```

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

```

1652 \def\bbl@redefine@long#1{%
1653   \edef\bbl@tempa{\bbl@stripslash#1}%
1654   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1655   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname{
1656     \@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_`.

```
1657 \def\bbl@redefineroast#1{%
1658   \edef\bbl@tempa{\bbl@stripslash#1}%
1659   \bbl@ifunset{\bbl@tempa\space}%
1660   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1661     \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1662   {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1663   \@namedef{\bbl@tempa\space}}
1664 \@onlypreamble\bbl@redefineroast
```

### 9.3 Hooks

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

```
1665 \bbl@trace{Hooks}
1666 \newcommand\AddBabelHook[3][{}{%
1667   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1668   \def\bbl@tempa##1,##3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1669   \expandafter\bbl@tempa\bbl@evargs,##3=,\@empty
1670   \bbl@ifunset{bbl@ev@#2@#3@#1}%
1671     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elt{#2}}}%
1672     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1673   \bbl@csarg\newcommand{ev@#2@#3@#1}{\bbl@tempb}}
1674 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1675 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1676 \def\bbl@usehooks#1#2{%
1677   \def\bbl@elt##1{%
1678     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@#2}}%
1679     \bbl@cs{ev@#1@}%
1680     \ifx\language\undefined\else % Test required for Plain (?)
1681       \def\bbl@elt##1{%
1682         \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1@#2}}%
1683         \bbl@cl{ev@#1@}%
1684       \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).

```
1685 \def\bbl@evargs{,% <- don't delete this comma
1686   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1687   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1688   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1689   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1690   beforestart=0,language=2}
```

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

The macro `\bbl@e@<language>` contains `\bbl@ensure{\<include>}{\<exclude>}{\<fontenc>}`, which in turn loops over the macros names in `\bbl@captionslist`, excluding (with the help of `\in@`) those in the exclude list. If the fontenc is given (and not `\relax`), the `\fontencoding` is also added. Then we loop over the include list, but if the macro already

contains \foreignlanguage, nothing is done. Note this macro (1) is not restricted to the preamble, and (2) changes are local.

```

1691 \bbl@trace{Defining babelensure}
1692 \newcommand\babelensure[2][{}]{% TODO - revise test files
1693   \AddBabelHook{babel-ensure}{afterextras}{%
1694     \ifcase\bbl@select@type
1695       \bbl@cl{e}%
1696     \fi}%
1697   \begingroup
1698     \let\bbl@ens@include\@empty
1699     \let\bbl@ens@exclude\@empty
1700     \def\bbl@ens@fontenc{\relax}%
1701     \def\bbl@tempb##1{%
1702       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1703     \edef\bbl@tempa{\bbl@tempb##1\@empty}%
1704     \def\bbl@tempb##1=##2\@{\@namedef{bbl@ens@##1}{##2}}%
1705     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1706     \def\bbl@tempc{\bbl@ensure}%
1707     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1708       \expandafter{\bbl@ens@include}}%
1709     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1710       \expandafter{\bbl@ens@exclude}}%
1711     \toks@\expandafter{\bbl@tempc}%
1712     \bbl@exp{%
1713   \endgroup
1714   \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1715 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1716   \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1717     \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1718       \edef##1{\noexpand\bbl@nocaption
1719         {\bbl@stripslash##1}{\language\name\bbl@stripslash##1}}%
1720     \fi
1721     \ifx##1\@empty\else
1722       \in@{##1}{#2}%
1723     \ifin@ \else
1724       \bbl@ifunset{bbl@ensure@\language\name}%
1725       {\bbl@exp{%
1726         \\DeclareRobustCommand\<bbl@ensure@\language\name>[1]{%
1727           \\foreignlanguage{\language\name}%
1728           {\ifx\relax#3\else
1729             \\fontencoding{#3}\\selectfont
1730           \fi
1731           #####1}}}%
1732       {}%
1733       \toks@\expandafter{##1}%
1734       \edef##1{%
1735         \bbl@csarg\noexpand{ensure@\language\name}%
1736         {\the\toks@}}%
1737     \fi
1738     \expandafter\bbl@tempb
1739   \fi}%
1740 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1741 \def\bbl@tempa##1{% elt for include list
1742   \ifx##1\@empty\else
1743     \bbl@csarg\in@{ensure@\language\name\expandafter}\expandafter{##1}%
1744   \ifin@ \else
1745     \bbl@tempb##1\@empty
1746   \fi

```

```

1747     \expandafter\bb1@tempa
1748     \fi}%
1749     \bb1@tempa#1\@empty}
1750 \def\bb1@captionslist{%
1751   \prefacename\refname\abstractname\bibname\chaptername\appendixname
1752   \contentsname\listfigurename\listtablename\indexname\figurename
1753   \tablename\partname\enc1name\ccname\headtoname\pagename\seename
1754   \alsoname\proofname\glossaryname}

```

## 9.4 Setting up language files

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

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

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

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

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

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

Finally we check `\originalTeX`.

```

1755 \bb1@trace{Macros for setting language files up}
1756 \def\bb1\ldfinit{% TODO. Merge into the next macro? Unused elsewhere
1757   \let\bb1@screset\@empty
1758   \let\BabelStrings\bb1@opt@string
1759   \let\BabelOptions\@empty
1760   \let\BabelLanguages\relax
1761   \ifx\originalTeX\@undefined
1762     \let\originalTeX\@empty
1763   \else
1764     \originalTeX
1765   \fi}
1766 \def\LdfInit#1#2{%
1767   \chardef\atcatcode=\catcode`\@
1768   \catcode`\@=11\relax
1769   \chardef\eqcatcode=\catcode`\=
1770   \catcode`\==12\relax
1771   \expandafter\if\expandafter\@backslashchar
1772     \expandafter\@car\string#2\@nil
1773     \ifx#2\@undefined\else
1774       \ldf@quit{#1}%
1775     \fi
1776   \else
1777     \expandafter\ifx\csname#2\endcsname\relax\else
1778       \ldf@quit{#1}%
1779     \fi
1780   \fi

```



```
1781 \bbl@ldfinit}
```

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

```
1782 \def\ldf@quit#1{%
1783 \expandafter\main@language\expandafter{#1}%
1784 \catcode`\@=\atcatcode \let\atcatcode\relax
1785 \catcode`\==\eqcatcode \let\eqcatcode\relax
1786 \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.

```
1787 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1788 \bbl@afterlang
1789 \let\bbl@afterlang\relax
1790 \let\BabelModifiers\relax
1791 \let\bbl@screset\relax}%
1792 \def\ldf@finish#1{%
1793 \ifx\loadlocalcfg\@undefined\else % For LaTeX 209
1794 \loadlocalcfg{#1}%
1795 \fi
1796 \bbl@afterldf{#1}%
1797 \expandafter\main@language\expandafter{#1}%
1798 \catcode`\@=\atcatcode \let\atcatcode\relax
1799 \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  $\LaTeX$ .

```
1800 \@onlypreamble\LdfInit
1801 \@onlypreamble\ldf@quit
1802 \@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.

```
1803 \def\main@language#1{%
1804 \def\bbl@main@language{#1}%
1805 \let\language\name\bbl@main@language % TODO. Set localname
1806 \bbl@id@assign
1807 \bbl@patterns{\language}}
```

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

```
1808 \def\bbl@beforestart{%
1809 \bbl@usehooks{beforestart}{}%
1810 \global\let\bbl@beforestart\relax}
1811 \AtBeginDocument{%
1812 \@nameuse{bbl@beforestart}%
1813 \if@files
1814 \providecommand\babel@aux[2]{}%
1815 \immediate\write\@mainaux{%
1816 \string\providecommand\string\babel@aux[2]{}%
1817 \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}}%

```

```

1818 \fi
1819 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1820 \ifbbl@single % must go after the line above.
1821 \renewcommand\selectlanguage[1]{}%
1822 \renewcommand\foreignlanguage[2]{#2}%
1823 \global\let\babel@aux\@gobbletwo % Also as flag
1824 \fi
1825 \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1826 \def\select@language@x#1{%
1827 \ifcase\bbl@select@type
1828 \bbl@ifsamestring\language#1\{\select@language{#1}}%
1829 \else
1830 \select@language{#1}%
1831 \fi}

```

## 9.5 Shorthands

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

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

```

1832 \bbl@trace{Shorhands}
1833 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1834 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1835 \bbl@ifunset{@sanitize}\{\bbl@add\@sanitize{\@makeother#1}}%
1836 \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1837 \begingroup
1838 \catcode`#1\active
1839 \nfss@catcodes
1840 \ifnum\catcode`#1=\active
1841 \endgroup
1842 \bbl@add\nfss@catcodes{\@makeother#1}%
1843 \else
1844 \endgroup
1845 \fi
1846 \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.

```

1847 \def\bbl@remove@special#1{%
1848 \begingroup
1849 \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1850 \else\noexpand##1\noexpand##2\fi}%
1851 \def\do{\x\do}%
1852 \def\@makeother{\x\@makeother}%
1853 \edef\x{\endgroup
1854 \def\noexpand\dospecials{\dospecials}%
1855 \expandafter\ifx\csname @sanitize\endcsname\relax\else
1856 \def\noexpand\@sanitize{\@sanitize}%
1857 \fi}%
1858 \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).

```
1859 \def\bbl@active@def#1#2#3#4{%
1860   \@namedef{#3#1}{%
1861     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1862       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1863     \else
1864       \bbl@afterfi\csname#2@sh@#1\endcsname
1865     \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.

```
1866   \long\@namedef{#3@arg#1}##1{%
1867     \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
1868       \bbl@afterelse\csname#4#1\endcsname##1%
1869     \else
1870       \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
1871     \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.

```
1872 \def\@initiate@active@char#1#2#3{%
1873   \bbl@ifunset{active@char\string#1}%
1874   {\bbl@withactive
1875     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1876   {}}

```

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`).

```
1877 \def\@initiate@active@char#1#2#3{%
1878   \bbl@csarg\edef\oricat@#2{\catcode`#2=\the\catcode`#2\relax}%
1879   \ifx#1\@undefined
1880     \bbl@csarg\edef\oridef@#2{\let\noexpand#1\noexpand\@undefined}%
1881   \else
1882     \bbl@csarg\let\oridef@#2#1%
1883     \bbl@csarg\edef\oridef@#2{%

```

```

1884 \let\noexpand#1%
1885 \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1886 \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`).

```

1887 \ifx#1#3\relax
1888 \expandafter\let\csname normal@char#2\endcsname#3%
1889 \else
1890 \bbl@info{Making #2 an active character}%
1891 \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1892 \@namedef{normal@char#2}{%
1893 \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1894 \else
1895 \@namedef{normal@char#2}{#3}%
1896 \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).

```

1897 \bbl@restoreactive{#2}%
1898 \AtBeginDocument{%
1899 \catcode`#2\active
1900 \if@filesw
1901 \immediate\write\@mainaux{\catcode`\string#2\active}%
1902 \fi}%
1903 \expandafter\bbl@add@special\csname#2\endcsname
1904 \catcode`#2\active
1905 \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⟩`).

```

1906 \let\bbl@tempa\@firstoftwo
1907 \if\string^#2%
1908 \def\bbl@tempa{\noexpand\textormath}%
1909 \else
1910 \ifx\bbl@mathnormal\@undefined\else
1911 \let\bbl@tempa\bbl@mathnormal
1912 \fi
1913 \fi
1914 \expandafter\edef\csname active@char#2\endcsname{%
1915 \bbl@tempa
1916 {\noexpand\if@safe@actives
1917 \noexpand\expandafter
1918 \expandafter\noexpand\csname normal@char#2\endcsname
1919 \noexpand\else
1920 \noexpand\expandafter

```

```

1921      \expandafter\noexpand\csname bbl@doactive#2\endcsname
1922      \noexpand\fi}%
1923      {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1924      \bbl@csarg\edef{doactive#2}{%
1925      \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!).

```

1926      \bbl@csarg\edef{active@#2}{%
1927      \noexpand\active@prefix\noexpand#1%
1928      \expandafter\noexpand\csname active@char#2\endcsname}%
1929      \bbl@csarg\edef{normal@#2}{%
1930      \noexpand\active@prefix\noexpand#1%
1931      \expandafter\noexpand\csname normal@char#2\endcsname}%
1932      \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.

```

1933      \bbl@active@def#2\user@group{user@active}{language@active}%
1934      \bbl@active@def#2\language@group{language@active}{system@active}%
1935      \bbl@active@def#2\system@group{system@active}{normal@char}%

```

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

```

1936      \expandafter\edef\csname\user@group @sh@#2@@\endcsname
1937      {\expandafter\noexpand\csname normal@char#2\endcsname}%
1938      \expandafter\edef\csname\user@group @sh@#2@\string\protect\endcsname
1939      {\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.

```

1940      \if\string'#2%
1941      \let\prim@s\bbl@prim@s
1942      \let\active@math@prime#1%
1943      \fi
1944      \bbl@usehooks{initiateactive}{\{#1\}{#2\}{#3\}}

```

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

```

1945      <<{*More package options}>> ≡
1946      \DeclareOption{math=active}{}
1947      \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1948      <</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*.

```

1949      \ifpackagewith{babel}{KeepShorthandsActive}%

```

```

1950 {\let\bbl@restoreactive\@gobble}%
1951 {\def\bbl@restoreactive#1{%
1952   \bbl@exp{%
1953     \\AfterBabelLanguage\\CurrentOption
1954     {\catcode`#1=\the\catcode`#1\relax}%
1955     \\AtEndOfPackage
1956     {\catcode`#1=\the\catcode`#1\relax}}}%
1957   \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.

```

1958 \def\bbl@sh@select#1#2{%
1959   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1960     \bbl@afterelse\bbl@scndcs
1961   \else
1962     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1963   \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.

```

1964 \begingroup
1965 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
1966 {\gdef\active@prefix#1{%
1967   \ifx\protect\@typeset@protect
1968   \else
1969     \ifx\protect\@unexpandable@protect
1970       \noexpand#1%
1971     \else
1972       \protect#1%
1973     \fi
1974     \expandafter\@gobble
1975   \fi}}
1976 {\gdef\active@prefix#1{%
1977   \ifincsname
1978     \string#1%
1979     \expandafter\@gobble
1980   \else
1981     \ifx\protect\@typeset@protect
1982     \else
1983       \ifx\protect\@unexpandable@protect
1984         \noexpand#1%
1985       \else
1986         \protect#1%
1987       \fi
1988       \expandafter\expandafter\expandafter\@gobble
1989     \fi
1990   \fi}}
1991 \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⟩`.

```

1992 \newif\if@safe@actives
1993 \@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.

```

1994 \def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}

```

`\bbl@activate` Both macros take one argument, like `\initiate@active@char`. The macro is used to change the definition of an active character to expand to `\active@char⟨char⟩` in the case of `\bbl@activate`, or `\normal@char⟨char⟩` in the case of `\bbl@deactivate`.

`\bbl@deactivate`

```

1995 \def\bbl@activate#1{%
1996   \bbl@withactive{\expandafter\let\expandafter}\#1%
1997   \csname bbl@active@\string#1\endcsname}
1998 \def\bbl@deactivate#1{%
1999   \bbl@withactive{\expandafter\let\expandafter}\#1%
2000   \csname bbl@normal@\string#1\endcsname}

```

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

`\bbl@scndcs`

```

2001 \def\bbl@firstcs#1#2{\csname#1\endcsname}
2002 \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.

```

2003 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
2004 \def\@decl@short#1#2#3\@nil#4{%
2005   \def\bbl@tempa{#3}%
2006   \ifx\bbl@tempa\@empty
2007     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
2008     \bbl@ifunset{#1@sh@\string#2@}{}%
2009     {\def\bbl@tempa{#4}%
2010      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
2011      \else
2012        \bbl@info
2013          {Redefining #1 shorthand \string#2\%
2014           in language \CurrentOption}%
2015        \fi}%
2016     \@namedef{#1@sh@\string#2@}{#4}%
2017   \else
2018     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
2019     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
2020     {\def\bbl@tempa{#4}%
2021      \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
2022      \else
2023        \bbl@info
2024          {Redefining #1 shorthand \string#2\string#3\%
2025           in language \CurrentOption}%
2026        \fi}%
2027     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2028   \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.

```

2029 \def\textormath{%
2030   \ifmmode
2031     \expandafter\@secondoftwo
2032   \else
2033     \expandafter\@firstoftwo
2034   \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’.

```

2035 \def\user@group{user}
2036 \def\language@group{english} % TODO. I don't like defaults
2037 \def\system@group{system}

```

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

```

2038 \def\useshorthands{%
2039   \@ifstar\bb1@usesh@s{\bb1@usesh@x{}}
2040 \def\bb1@usesh@s#1{%
2041   \bb1@usesh@x
2042     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb1@activate{#1}}}%
2043     {#1}}
2044 \def\bb1@usesh@x#1#2{%
2045   \bb1@ifshorthand{#2}%
2046     {\def\user@group{user}%
2047       \initiate@active@char{#2}%
2048       #1%
2049       \bb1@activate{#2}}%
2050     {\bb1@error
2051       {Cannot declare a shorthand turned off (\string#2)}
2052       {Sorry, but you cannot use shorthands which have been\\%
2053         turned off in the package options}}}

```

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

```

2054 \def\user@language@group{user@\language@group}
2055 \def\bb1@set@user@generic#1#2{%
2056   \bb1@ifunset{user@generic@active#1}%
2057     {\bb1@active@def#1\user@language@group{user@active}{user@generic@active}%
2058       \bb1@active@def#1\user@group{user@generic@active}{language@active}%
2059       \expandafter\edef\csname#2sh@#1@@\endcsname{%
2060         \expandafter\noexpand\csname normal@char#1\endcsname}%
2061       \expandafter\edef\csname#2sh@#1@\string\protect@\endcsname{%
2062         \expandafter\noexpand\csname user@active#1\endcsname}}%
2063   \@empty}
2064 \newcommand\defineshorthand[3][user]{%
2065   \edef\bb1@tempa{\zap@space#1 \@empty}%
2066   \bb1@for\bb1@tempb\bb1@tempa{%
2067     \if*\expandafter\@car\bb1@tempb\@nil

```



```

2068 \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
2069 \@expandtwoargs
2070 \bbl@set@user@generic{\expandafter\string\car#2\@nil}\bbl@tempb
2071 \fi
2072 \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 [TODO. Unclear].

```

2073 \def\languageshorthands#1{\def\language@group{#1}}

```

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

```

2074 \def\aliasshorthand#1#2{%
2075 \bbl@ifshorthand{#2}%
2076 {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2077 \ifx\document\@notprerr
2078 \@notshorthand{#2}%
2079 \else
2080 \initiate@active@char{#2}%
2081 \expandafter\let\csname active@char\string#2\expandafter\endcsname
2082 \csname active@char\string#1\endcsname
2083 \expandafter\let\csname normal@char\string#2\expandafter\endcsname
2084 \csname normal@char\string#1\endcsname
2085 \bbl@activate{#2}%
2086 \fi
2087 \fi}%
2088 {\bbl@error
2089 {Cannot declare a shorthand turned off (\string#2)}
2090 {Sorry, but you cannot use shorthands which have been\\%
2091 turned off in the package options}}}

```

`\@notshorthand`

```

2092 \def\@notshorthand#1{%
2093 \bbl@error{%
2094 The character '\string #1' should be made a shorthand character;\\%
2095 add the command \string\usesshorthands\string{#1\string} to
2096 the preamble.\\%
2097 I will ignore your instruction}%
2098 {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.

```

2099 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2100 \DeclareRobustCommand*\shorthandoff{%
2101 \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2102 \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.

```

2103 \def\bb@switch@sh#1#2{%
2104   \ifx#2\@nnil\else
2105     \bb@ifunset{bb@active@\string#2}%
2106     {\bb@error
2107       {I cannot switch '\string#2' on or off--not a shorthand}%
2108       {This character is not a shorthand. Maybe you made\\%
2109         a typing mistake? I will ignore your instruction}}%
2110     {\ifcase#1%
2111       \catcode`#2\relax
2112       \or
2113       \catcode`#2\active
2114       \or
2115       \csname bb@oricat@\string#2\endcsname
2116       \csname bb@oridef@\string#2\endcsname
2117       \fi}%
2118     \bb@afterfi\bb@switch@sh#1%
2119   \fi}

```

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

```

2120 \def\babelshorthand{\active@prefix\babelshorthand\bb@putsh}
2121 \def\bb@putsh#1{%
2122   \bb@ifunset{bb@active@\string#1}%
2123   {\bb@putsh@i#1\@empty\@nnil}%
2124   {\csname bb@active@\string#1\endcsname}}
2125 \def\bb@putsh@i#1#2\@nnil{%
2126   \csname\language\@sh@\string#1@%
2127     \ifx\@empty#2\else\string#2@\fi\endcsname}
2128 \ifx\bb@opt@shorthands\@nnil\else
2129   \let\bb@s@initiate@active@char\initiate@active@char
2130   \def\initiate@active@char#1{%
2131     \bb@ifshorthand{#1}{\bb@s@initiate@active@char{#1}}{}}
2132   \let\bb@s@switch@sh\bb@switch@sh
2133   \def\bb@switch@sh#1#2{%
2134     \ifx#2\@nnil\else
2135       \bb@afterfi
2136       \bb@ifshorthand{#2}{\bb@s@switch@sh#1{#2}}{\bb@switch@sh#1}%
2137     \fi}
2138   \let\bb@s@activate\bb@activate
2139   \def\bb@activate#1{%
2140     \bb@ifshorthand{#1}{\bb@s@activate{#1}}{}}
2141   \let\bb@s@deactivate\bb@deactivate
2142   \def\bb@deactivate#1{%
2143     \bb@ifshorthand{#1}{\bb@s@deactivate{#1}}{}}
2144 \fi

```

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

```

2145 \newcommand\ifbabelshorthand[3]{\bb@ifunset{bb@active@\string#1}{#3}{#2}}

```

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

```

2146 \def\bbl@prim@s{%
2147   \prime\futurelet\@let@token\bbl@pr@m@s}
2148 \def\bbl@if@primes#1#2{%
2149   \ifx#1\@let@token
2150     \expandafter\@firstoftwo
2151   \else\ifx#2\@let@token
2152     \bbl@afterelse\expandafter\@firstoftwo
2153   \else
2154     \bbl@afterfi\expandafter\@secondoftwo
2155   \fi\fi}
2156 \begingroup
2157   \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
2158   \catcode`\'=12 \catcode`\"=\active \lccode`\"=\'
2159   \lowercase{%
2160     \gdef\bbl@pr@m@s{%
2161       \bbl@if@primes"%
2162         \pr@@s
2163         {\bbl@if@primes*\^{\pr@@t\egroup}}}}
2164 \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).

```

2165 \initiate@active@char{~}
2166 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2167 \bbl@activate{~}

```

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

```

2168 \expandafter\def\csname OT1dqpos\endcsname{127}
2169 \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

```

2170 \ifx\f@encoding\@undefined
2171   \def\f@encoding{OT1}
2172 \fi

```

## 9.6 Language attributes

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

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

```

2173 \bbl@trace{Language attributes}
2174 \newcommand\languageattribute[2]{%
2175   \def\bbl@tempc{#1}%
2176   \bbl@fixname\bbl@tempc
2177   \bbl@iflanguage\bbl@tempc{%
2178     \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.

```

2179 \if\bbl@known@attrs\@undefined
2180 \in@false
2181 \else
2182 \bbl@xin@{\bbl@tempc-##1,}{,\bbl@known@attrs,}%
2183 \fi
2184 \ifin@
2185 \bbl@warning{%
2186 You have more than once selected the attribute '##1'\%
2187 for language #1. Reported}%
2188 \else

```

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

```

2189 \bbl@exp{%
2190 \\\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
2191 \edef\bbl@tempa{\bbl@tempc-##1}%
2192 \expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes%
2193 {\csname\bbl@tempc @attr##1\endcsname}%
2194 {\@attrerr{\bbl@tempc}{##1}}%
2195 \fi}}
2196 \@onlypreamble\languageattribute

```

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

```

2197 \newcommand*{\@attrerr}[2]{%
2198 \bbl@error
2199 {The attribute #2 is unknown for language #1.}%
2200 {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}`.

```

2201 \def\bbl@declare@ttribute#1#2#3{%
2202 \bbl@xin@{#2,}{,\BabelModifiers,}%
2203 \ifin@
2204 \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2205 \fi
2206 \bbl@add@list\bbl@attributes{#1-#2}%
2207 \expandafter\def\csname#1@attr#2\endcsname{#3}}

```

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

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

First we need to find out if any attributes were set; if not we're done. Then we need to check the list of known attributes. When we're this far `\ifin@` has a value indicating if the attribute in question was set or not. Just to be safe the code to be executed is 'thrown over the `\fi`'.

```

2208 \def\bbl@ifattributeset#1#2#3#4{%

```

```

2209 \ifx\bb1@known@attribs\@undefined
2210 \in@false
2211 \else
2212 \bb1@xin@{,#1-#2,}{,\bb1@known@attribs,}%
2213 \fi
2214 \ifin@
2215 \bb1@afterelse#3%
2216 \else
2217 \bb1@afterfi#4%
2218 \fi
2219 }

```

`\bb1@ifknown@ttrib` An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the  $\TeX$ -code to be executed when the attribute is known and the  $\TeX$ -code to be executed otherwise. We first assume the attribute is unknown. Then we loop over the list of known attributes, trying to find a match. When a match is found the definition of `\bb1@tempa` is changed. Finally we execute `\bb1@tempa`.

```

2220 \def\bb1@ifknown@ttrib#1#2{%
2221 \let\bb1@tempa\@secondoftwo
2222 \bb1@loopx\bb1@tempb{#2}{%
2223 \expandafter\in@expandafter{\expandafter,\bb1@tempb,}{,#1,}%
2224 \ifin@
2225 \let\bb1@tempa\@firstoftwo
2226 \else
2227 \fi}%
2228 \bb1@tempa
2229 }

```

`\bb1@clear@ttribs` This macro removes all the attribute code from  $\LaTeX$ 's memory at `\begin{document}` time (if any is present).

```

2230 \def\bb1@clear@ttribs{%
2231 \ifx\bb1@attributes\@undefined\else
2232 \bb1@loopx\bb1@tempa{\bb1@attributes}{%
2233 \expandafter\bb1@clear@ttrib\bb1@tempa.
2234 }%
2235 \let\bb1@attributes\@undefined
2236 \fi}
2237 \def\bb1@clear@ttrib#1-#2.{%
2238 \expandafter\let\csname#1@attr#2\endcsname\@undefined}
2239 \AtBeginDocument{\bb1@clear@ttribs}

```

## 9.7 Support for saving macro definitions

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

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

`\babel@beginsave`

```

2240 \bb1@trace{Macros for saving definitions}
2241 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```
2242 \newcount\babel@savecnt
2243 \babel@beginsave
```

`\babel@save` The macro `\babel@save⟨curname⟩` saves the current meaning of the control sequence `⟨curname⟩` to `\originalTeX`<sup>31</sup>. To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to `\originalTeX` and the counter is incremented. The macro `\babel@savevariable⟨variable⟩` saves the value of the variable. `⟨variable⟩` can be anything allowed after the `\the` primitive.

```
2244 \def\babel@save#1{%
2245   \expandafter\let\curname \babel@number\babel@savecnt\endcurname#1\relax
2246   \toks@\expandafter{\originalTeX\let#1=}
2247   \bbl@exp{%
2248     \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}
2249   \advance\babel@savecnt\@ne
2250 \def\babel@savevariable#1{%
2251   \toks@\expandafter{\originalTeX #1=}
2252   \bbl@exp{\def\\originalTeX{\the\toks@the#1\relax}}}
```

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

```
2253 \def\bbl@frenchspacing{%
2254   \ifnum\the\sffcode\`.\=@m
2255     \let\bbl@nonfrenchspacing\relax
2256   \else
2257     \frenchspacing
2258     \let\bbl@nonfrenchspacing\nonfrenchspacing
2259   \fi
2260 \let\bbl@nonfrenchspacing\nonfrenchspacing
```

## 9.8 Short tags

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

```
2261 \bbl@trace{Short tags}
2262 \def\babeltags#1{%
2263   \edef\bbl@tempa{\zap@space#1 \@empty}%
2264   \def\bbl@tempb##1=##2\@{
2265     \edef\bbl@tempc{%
2266       \noexpand\newcommand
2267       \expandafter\noexpand\curname ##1\endcurname{%
2268         \noexpand\protect
2269         \expandafter\noexpand\curname otherlanguage*\endcurname{##2}}
2270       \noexpand\newcommand
2271       \expandafter\noexpand\curname text##1\endcurname{%
2272         \noexpand\foreignlanguage{##2}}
2273     \bbl@tempc}%
2274   \bbl@for\bbl@tempa\bbl@tempa{%
2275     \expandafter\bbl@tempb\bbl@tempa\@{}}
```

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

## 9.9 Hyphens

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

```

2276 \bbl@trace{Hyphens}
2277 \@onlypreamble\babelhyphenation
2278 \AtEndOfPackage{%
2279   \newcommand\babelhyphenation[2][\@empty]{%
2280     \ifx\bbl@hyphenation@ \relax
2281       \let\bbl@hyphenation@ \@empty
2282     \fi
2283     \ifx\bbl@hyphlist \@empty \else
2284       \bbl@warning{%
2285         You must not intermingle \string\selectlanguage\space and\%
2286         \string\babelhyphenation\space or some exceptions will not\%
2287         be taken into account. Reported}%
2288       \fi
2289       \ifx \@empty #1%
2290         \protected@edef\bbl@hyphenation@{\bbl@hyphenation@ \space #2}%
2291       \else
2292         \bbl@vforeach{#1}{%
2293           \def\bbl@tempa{##1}%
2294           \bbl@fixname\bbl@tempa
2295           \bbl@iflanguage\bbl@tempa{%
2296             \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2297               \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
2298               \@empty
2299               {\csname bbl@hyphenation@\bbl@tempa\endcsname \space}%
2300               #2}}}%
2301         \fi}}

```

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

```

2302 \def\bbl@allowhyphens{\ifvmode \else \nobreak \hskip \zskip \fi}
2303 \def\bbl@t@one{T1}
2304 \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`.

```

2305 \newcommand\babellnullhyphen{\char\hyphenchar\font}
2306 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
2307 \def\bbl@hyphen{%
2308   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
2309 \def\bbl@hyphen@i#1#2{%
2310   \bbl@ifunset{bbl@hy@#1#2 \@empty}%
2311   {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{ }{#2}}}%
2312   {\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.

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

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.

```

2313 \def\bbl@usehyphen#1{%
2314   \leavevmode
2315   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2316   \nobreak\hskip\z@skip}
2317 \def\bbl@usehyphen#1{%
2318   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

2319 \def\bbl@hyphenchar{%
2320   \ifnum\hyphenchar\font=\m@ne
2321     \babeinullhyphen
2322   \else
2323     \char\hyphenchar\font
2324   \fi}

```

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

```

2325 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2326 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2327 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2328 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2329 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
2330 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
2331 \def\bbl@hy@repeat{%
2332   \bbl@usehyphen{%
2333     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2334 \def\bbl@hy@repeat{%
2335   \bbl@usehyphen{%
2336     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2337 \def\bbl@hy@empty{\hskip\z@skip}
2338 \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.

```

2339 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#1}\bbl@allowhyphens}

```

## 9.10 Multiencoding strings

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

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

```

2340 \bbl@trace{Multiencoding strings}
2341 \def\bbl@tglobal#1{\global\let#1#1}
2342 \def\bbl@recatcode#1{% TODO. Used only once?
2343   \@tempcnta="7F
2344   \def\bbl@tempa{%
2345     \ifnum\@tempcnta>"FF\else
2346       \catcode\@tempcnta=#1\relax
2347       \advance\@tempcnta@\ne
2348     \expandafter\bbl@tempa

```



```

2349   \fi}%
2350   \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).

```

2351 \@ifpackagewith{babel}{nocase}%
2352   {\let\bbl@patchuclc\relax}%
2353   {\def\bbl@patchuclc{%
2354     \global\let\bbl@patchuclc\relax
2355     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
2356     \gdef\bbl@uclc##1{%
2357       \let\bbl@encoded\bbl@encoded@uclc
2358       \bbl@ifunset{\language @bbl@uclc}% and resumes it
2359       {##1}%
2360       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
2361         \csname\language @bbl@uclc\endcsname}%
2362       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
2363     \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
2364     \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}%
2365 <<More package options>> ≡
2366 \DeclareOption{nocase}{}
2367 <</More package options>>

```

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

```

2368 <<More package options>> ≡
2369 \let\bbl@opt@strings\@nnil % accept strings=value
2370 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2371 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2372 \def\BabelStringsDefault{generic}
2373 <</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.

```

2374 \@onlypreamble\StartBabelCommands
2375 \def\StartBabelCommands{%
2376   \begingroup
2377   \bbl@recatcode{11}%
2378   <<Macros local to BabelCommands>>
2379   \def\bbl@provstring##1##2{%
2380     \providecommand##1{##2}%
2381     \bbl@tglobal##1}%
2382   \global\let\bbl@scafter\@empty
2383   \let\StartBabelCommands\bbl@startcmds
2384   \ifx\BabelLanguages\relax
2385     \let\BabelLanguages\CurrentOption
2386   \fi

```

```

2387 \begingroup
2388 \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2389 \StartBabelCommands}
2390 \def\bbl@startcmds{%
2391   \ifx\bbl@screset\@nnil\else
2392     \bbl@usehooks{stopcommands}{}%
2393   \fi
2394 \endgroup
2395 \begingroup
2396 \@ifstar
2397   {\ifx\bbl@opt@strings\@nnil
2398     \let\bbl@opt@strings\BabelStringsDefault
2399     \fi
2400     \bbl@startcmds@i}%
2401   \bbl@startcmds@i}
2402 \def\bbl@startcmds@i#1#2{%
2403   \edef\bbl@L{\zap@space#1 \@empty}%
2404   \edef\bbl@G{\zap@space#2 \@empty}%
2405   \bbl@startcmds@ii}
2406 \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.

```

2407 \newcommand\bbl@startcmds@ii[1][\@empty]{%
2408   \let\SetString@gobbletwo
2409   \let\bbl@stringdef@gobbletwo
2410   \let\AfterBabelCommands@gobble
2411   \ifx\@empty#1%
2412     \def\bbl@sc@label{generic}%
2413     \def\bbl@encstring##1##2{%
2414       \ProvideTextCommandDefault##1{##2}%
2415       \bbl@toglobal##1%
2416       \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
2417     \let\bbl@sctest\in@true
2418   \else
2419     \let\bbl@sc@charset\space % <- zapped below
2420     \let\bbl@sc@fontenc\space % <- " "
2421     \def\bbl@tempa##1=##2\@nil{%
2422       \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
2423     \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
2424     \def\bbl@tempa##1 ##2{% space -> comma
2425       ##1%
2426       \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
2427     \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
2428     \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
2429     \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
2430     \def\bbl@encstring##1##2{%
2431       \bbl@foreach\bbl@sc@fontenc{%
2432         \bbl@ifunset{T@###1}%
2433       }%

```

```

2434         {\ProvideTextCommand###1{####1}{##2}%
2435         \bbl@toglobal##1%
2436         \expandafter
2437         \bbl@toglobal\csname####1\string##1\endcsname}}}%
2438     \def\bbl@sctest{%
2439         \bbl@xin@{\,\bbl@opt@strings,}{\,\bbl@sc@label,\bbl@sc@fontenc,}}%
2440 \fi
2441 \ifx\bbl@opt@strings\@nnil      % ie, no strings key -> defaults
2442 \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
2443     \let\AfterBabelCommands\bbl@aftercmds
2444     \let\SetString\bbl@setstring
2445     \let\bbl@stringdef\bbl@encstring
2446 \else      % ie, strings=value
2447 \bbl@sctest
2448 \ifin@
2449     \let\AfterBabelCommands\bbl@aftercmds
2450     \let\SetString\bbl@setstring
2451     \let\bbl@stringdef\bbl@provstring
2452 \fi\fi\fi
2453 \bbl@scswitch
2454 \ifx\bbl@G\@empty
2455     \def\SetString##1##2{%
2456         \bbl@error{Missing group for string \string##1}%
2457         {You must assign strings to some category, typically\\%
2458         captions or extras, but you set none}}%
2459 \fi
2460 \ifx\@empty#1%
2461     \bbl@usehooks{defaultcommands}}}%
2462 \else
2463     \@expandtwoargs
2464     \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}}%
2465 \fi}

```

There are two versions of `\bbl@scswitch`. The first version is used when `ldfs` are read, and it makes sure `\langle group \rangle \langle language \rangle` is reset, but only once (`\bbl@screset` is used to keep track of this). The second version is used in the preamble and packages loaded after `babel` and does nothing.

The macro `\bbl@forlang` loops `\bbl@L` but its body is executed only if the value is in `\BabelLanguages` (inside `babel`) or `\date \langle language \rangle` is defined (after `babel` has been loaded). There are also two version of `\bbl@forlang`. The first one skips the current iteration if the language is not in `\BabelLanguages` (used in `ldfs`), and the second one skips undefined languages (after `babel` has been loaded).

```

2466 \def\bbl@forlang#1#2{%
2467     \bbl@for#1\bbl@L{%
2468         \bbl@xin@{,#1,}{\,\BabelLanguages,}%
2469         \ifin@#2\relax\fi}}
2470 \def\bbl@scswitch{%
2471     \bbl@forlang\bbl@tempa{%
2472         \ifx\bbl@G\@empty\else
2473             \ifx\SetString\@gobbletwo\else
2474                 \edef\bbl@GL{\bbl@G\bbl@tempa}%
2475                 \bbl@xin@{\,\bbl@GL,}{\,\bbl@screset,}%
2476             \ifin@\else
2477                 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
2478                 \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
2479             \fi
2480         \fi
2481     \fi}}

```

```

2482 \AtEndOfPackage{%
2483   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\#2}}}%
2484   \let\bbl@scswitch\relax}
2485 \@onlypreamble\EndBabelCommands
2486 \def\EndBabelCommands{%
2487   \bbl@usehooks{stopcommands}{}%
2488   \endgroup
2489   \endgroup
2490   \bbl@scafter}
2491 \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.

```

2492 \def\bbl@setstring#1#2{%
2493   \bbl@forlang\bbl@tempa{%
2494     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2495     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2496     {\global\expandafter % TODO - con \bbl@exp ?
2497      \bbl@add\csname\bbl@G\bbl@tempa\expandafter\endcsname\expandafter
2498       {\expandafter\bbl@scset\expandafter#1\csname\bbl@LC\endcsname}}}%
2499     {}}%
2500   \def\BabelString{\#2}%
2501   \bbl@usehooks{stringprocess}{}%
2502   \expandafter\bbl@stringdef
2503   \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.

```

2504 \ifx\bbl@opt@strings\relax
2505   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2506   \bbl@patchuclc
2507   \let\bbl@encoded\relax
2508   \def\bbl@encoded@uclc#1{%
2509     \@inmathwarn#1%
2510     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2511       \expandafter\ifx\csname ?\string#1\endcsname\relax
2512         \TextSymbolUnavailable#1%
2513       \else
2514         \csname ?\string#1\endcsname
2515       \fi
2516     \else
2517       \csname\cf@encoding\string#1\endcsname
2518     \fi}
2519 \else
2520   \def\bbl@scset#1#2{\def#1{\#2}}
2521 \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.

```

2522 <<*Macros local to BabelCommands>> ≡
2523 \def\SetStringLoop##1##2{%
2524   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2525   \count@ \z@
2526   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2527     \advance\count@ \@ne
2528     \toks@\expandafter{\bbl@tempa}%
2529     \bbl@exp{%
2530       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2531       \count@=\the\count@\relax}}}%
2532 <</Macros local to BabelCommands>>

```

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

```

2533 \def\bbl@aftercmds#1{%
2534   \toks@\expandafter{\bbl@scafter#1}%
2535   \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.

```

2536 <<*Macros local to BabelCommands>> ≡
2537 \newcommand\SetCase[3][]{%
2538   \bbl@patchuclc
2539   \bbl@forlang\bbl@tempa{%
2540     \expandafter\bbl@encstring
2541     \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
2542     \expandafter\bbl@encstring
2543     \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2544     \expandafter\bbl@encstring
2545     \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2546 <</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.

```

2547 <<*Macros local to BabelCommands>> ≡
2548 \newcommand\SetHyphenMap[1]{%
2549   \bbl@forlang\bbl@tempa{%
2550     \expandafter\bbl@stringdef
2551     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
2552 <</Macros local to BabelCommands>>

```

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

```

2553 \newcommand\BabelLower[2]{% one to one.
2554   \ifnum\lccode#1=#2\else
2555     \babel@savevariable{\lccode#1}%
2556     \lccode#1=#2\relax
2557   \fi}
2558 \newcommand\BabelLowerMM[4]{% many-to-many
2559   \@tempcnta=#1\relax
2560   \@tempcntb=#4\relax
2561   \def\bbl@tempa{%
2562     \ifnum\@tempcnta>#2\else
2563       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2564       \advance\@tempcnta#3\relax
2565       \advance\@tempcntb#3\relax
2566       \expandafter\bbl@tempa

```

```

2567 \fi}%
2568 \bbl@tempa}
2569 \newcommand\BabelLowerM0[4]{% many-to-one
2570 \@tempcnta=#1\relax
2571 \def\bbl@tempa{%
2572 \ifnum\@tempcnta>#2\else
2573 \expandafter\BabelLower{\the\@tempcnta}{#4}%
2574 \advance\@tempcnta#3
2575 \expandafter\bbl@tempa
2576 \fi}%
2577 \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

2578 <<*More package options>> ≡
2579 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
2580 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
2581 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
2582 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
2583 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
2584 <</More package options>>

```

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

```

2585 \AtEndOfPackage{%
2586 \ifx\bbl@opt@hyphenmap\undefined
2587 \bbl@xin@{,}{\bbl@language@opts}%
2588 \chardef\bbl@opt@hyphenmap\ifin4\else\@ne\fi
2589 \fi}

```

## 9.11 Macros common to a number of languages

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

```

2590 \bbl@trace{Macros related to glyphs}
2591 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2592 \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2593 \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.

```

2594 \def\save@sf@q#1{\leavevmode
2595 \begingroup
2596 \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2597 \endgroup}

```

## 9.12 Making glyphs available

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

### 9.12.1 Quotation marks

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

```

2598 \ProvideTextCommand{\quotedblbase}{OT1}{%
2599 \save@sf@q{\set@low@box{\textquotedblright\}%
2600 \box\z@\kern-.04em\bbl@allowhyphens}}

```

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

```
2601 \ProvideTextCommandDefault{\quotedblbase}{%
2602   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2603 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2604   \save@sf@q{\set@low@box{\textquoteright\}%
2605     \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.

```
2606 \ProvideTextCommandDefault{\quotesinglbase}{%
2607   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2608 \ProvideTextCommand{\guillemetleft}{OT1}{%
2609   \ifmmode
2610     \ll
2611   \else
2612     \save@sf@q{\nobreak
2613       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bb1@allowhyphens}%
2614   \fi}
2615 \ProvideTextCommand{\guillemetright}{OT1}{%
2616   \ifmmode
2617     \gg
2618   \else
2619     \save@sf@q{\nobreak
2620       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bb1@allowhyphens}%
2621   \fi}
2622 \ProvideTextCommand{\guillemotleft}{OT1}{%
2623   \ifmmode
2624     \ll
2625   \else
2626     \save@sf@q{\nobreak
2627       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bb1@allowhyphens}%
2628   \fi}
2629 \ProvideTextCommand{\guillemotright}{OT1}{%
2630   \ifmmode
2631     \gg
2632   \else
2633     \save@sf@q{\nobreak
2634       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bb1@allowhyphens}%
2635   \fi}
```

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

```
2636 \ProvideTextCommandDefault{\guillemetleft}{%
2637   \UseTextSymbol{OT1}{\guillemetleft}}
2638 \ProvideTextCommandDefault{\guillemetright}{%
2639   \UseTextSymbol{OT1}{\guillemetright}}
2640 \ProvideTextCommandDefault{\guillemotleft}{%
2641   \UseTextSymbol{OT1}{\guillemotleft}}
2642 \ProvideTextCommandDefault{\guillemotright}{%
2643   \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
2644 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2645   \ifmmode
2646     <%
2647   \else
2648     \save@sf@q{\nobreak
2649       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2650   \fi}
2651 \ProvideTextCommand{\guilsinglright}{OT1}{%
2652   \ifmmode
2653     >%
2654   \else
2655     \save@sf@q{\nobreak
2656       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2657   \fi}
```

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

```
2658 \ProvideTextCommandDefault{\guilsinglleft}{%
2659   \UseTextSymbol{OT1}{\guilsinglleft}}
2660 \ProvideTextCommandDefault{\guilsinglright}{%
2661   \UseTextSymbol{OT1}{\guilsinglright}}
```

### 9.12.2 Letters

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

```
2662 \DeclareTextCommand{\ij}{OT1}{%
2663   i\kern-0.02em\bbl@allowhyphens j}
2664 \DeclareTextCommand{\IJ}{OT1}{%
2665   I\kern-0.02em\bbl@allowhyphens J}
2666 \DeclareTextCommand{\ij}{T1}{\char188}
2667 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2668 \ProvideTextCommandDefault{\ij}{%
2669   \UseTextSymbol{OT1}{\ij}}
2670 \ProvideTextCommandDefault{\IJ}{%
2671   \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).

```
2672 \def\crrtic@{\hrule height0.1ex width0.3em}
2673 \def\crrtic@{\hrule height0.1ex width0.33em}
2674 \def\ddj@{%
2675   \setbox0\hbox{d}\dimen@=\ht0
2676   \advance\dimen@1ex
2677   \dimen@.45\dimen@
2678   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2679   \advance\dimen@ii.5ex
2680   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2681 \def\DDJ@{%
2682   \setbox0\hbox{D}\dimen@=.55\ht0
2683   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
```



```

2684 \advance\dimen@ii.15ex % correction for the dash position
2685 \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2686 \dimen\thr@@\expandafter\rem\pt\the\fontdimen7\font\dimen@
2687 \leavevmode\rlap{\raise\dimen@hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2688 %
2689 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2690 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2691 \ProvideTextCommandDefault{\dj}{%
2692 \UseTextSymbol{OT1}{\dj}}
2693 \ProvideTextCommandDefault{\DJ}{%
2694 \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.

```

2695 \DeclareTextCommand{\SS}{OT1}{SS}
2696 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}

```

### 9.12.3 Shorthands for quotation marks

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

\glq The ‘german’ single quotes.

```

\grq 2697 \ProvideTextCommandDefault{\glq}{%
2698 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

```

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

```

2699 \ProvideTextCommand{\grq}{T1}{%
2700 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2701 \ProvideTextCommand{\grq}{TU}{%
2702 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2703 \ProvideTextCommand{\grq}{OT1}{%
2704 \save@sf@q{\kern-.0125em
2705 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2706 \kern.07em\relax}}
2707 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}

```

\glqq The ‘german’ double quotes.

```

\grqq 2708 \ProvideTextCommandDefault{\glqq}{%
2709 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

2710 \ProvideTextCommand{\grqq}{T1}{%
2711 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2712 \ProvideTextCommand{\grqq}{TU}{%
2713 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2714 \ProvideTextCommand{\grqq}{OT1}{%
2715 \save@sf@q{\kern-.07em
2716 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2717 \kern.07em\relax}}
2718 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}

```

`\flq` The ‘french’ single guillemets.  
`\frq` 2719 \ProvideTextCommandDefault{\flq}{%  
 2720 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}  
 2721 \ProvideTextCommandDefault{\frq}{%  
 2722 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

`\flqq` The ‘french’ double guillemets.  
`\frqq` 2723 \ProvideTextCommandDefault{\flqq}{%  
 2724 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}  
 2725 \ProvideTextCommandDefault{\frqq}{%  
 2726 \textormath{\guillemetright}{\mbox{\guillemetright}}}

#### 9.12.4 Umlauts and tremas

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

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

```
2727 \def\umlauthigh{%
2728   \def\bbl@umlauta##1{\leavevmode\bgroup%
2729     \expandafter\accent\csname\fontencoding dpos\endcsname
2730     ##1\bbl@allowhyphens\egroup}%
2731   \let\bbl@umlaute\bbl@umlauta}
2732 \def\umlautlow{%
2733   \def\bbl@umlauta{\protect\lower@umlaut}}
2734 \def\umlautelow{%
2735   \def\bbl@umlaute{\protect\lower@umlaut}}
2736 \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.

```
2737 \expandafter\ifx\csname U@D\endcsname\relax
2738   \csname newdimen\endcsname\U@D
2739 \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.

```
2740 \def\lower@umlaut#1{%
2741   \leavevmode\bgroup
2742   \U@D 1ex%
2743   {\setbox\z@\hbox{%
2744     \expandafter\char\csname\fontencoding dpos\endcsname}%
2745     \dimen@ -.45ex\advance\dimen@\ht\z@
2746     \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2747   \expandafter\accent\csname\fontencoding dpos\endcsname
```

```

2748 \fontdimen5\font\U@D #1%
2749 \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).

```

2750 \AtBeginDocument{%
2751 \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2752 \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2753 \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2754 \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{i}}%
2755 \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
2756 \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
2757 \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
2758 \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2759 \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2760 \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
2761 \DeclareTextCompositeCommand{\}{OT1}{U}{\bbl@umlauta{U}}%

```

Finally, make sure the default hyphenrules are defined (even if empty). For internal use, another empty `\language` is defined. Currently used in Amharic.

```

2762 \ifx\l@english\@undefined
2763 \chardef\l@english\z@
2764 \fi
2765 % The following is used to cancel rules in ini files (see Amharic).
2766 \ifx\l@babelnohyphens\@undefined
2767 \newlanguage\l@babelnohyphens
2768 \fi

```

## 9.13 Layout

### Work in progress.

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

```

2769 \bbl@trace{Bidi layout}
2770 \providecommand\IfBabelLayout[3]{#3}%
2771 \newcommand\BabelPatchSection[1]{%
2772 \@ifundefined{#1}{%
2773 \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2774 \@namedef{#1}{%
2775 \@ifstar{\bbl@presec@s{#1}}%
2776 {\@dblarg{\bbl@presec@x{#1}}}}%
2777 \def\bbl@presec@x#1[#2]#3{%
2778 \bbl@exp{%
2779 \\\select@language@x{\bbl@main@language}%
2780 \\\bbl@cs{sspre@#1}%
2781 \\\bbl@cs{ss@#1}%
2782 [\\foreignlanguage{\language}{\unexpanded{#2}}}%
2783 {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2784 \\\select@language@x{\language}}%
2785 \def\bbl@presec@s#1#2{%
2786 \bbl@exp{%
2787 \\\select@language@x{\bbl@main@language}%

```

```

2788   \\\bbl@cs{sspre@#1}%
2789   \\\bbl@cs{ss@#1}*%
2790   {\foreignlanguage{\language}{\unexpanded{#2}}}%
2791   \select@language@x{\language}}}%
2792 \IfBabelLayout{sectioning}%
2793   {\BabelPatchSection{part}%
2794    \BabelPatchSection{chapter}%
2795    \BabelPatchSection{section}%
2796    \BabelPatchSection{subsection}%
2797    \BabelPatchSection{subsubsection}%
2798    \BabelPatchSection{paragraph}%
2799    \BabelPatchSection{subparagraph}%
2800    \def\babel@toc#1{%
2801      \select@language@x{\bbl@main@language}}}%
2802 \IfBabelLayout{captions}%
2803   {\BabelPatchSection{caption}}}%

```

## 9.14 Load engine specific macros

```

2804 \bbl@trace{Input engine specific macros}
2805 \ifcase\bbl@engine
2806   \input txtbabel.def
2807 \or
2808   \input luababel.def
2809 \or
2810   \input xebabel.def
2811 \fi

```

## 9.15 Creating and modifying languages

`\babelprovide` is a general purpose tool for creating and modifying languages. It creates the language infrastructure, and loads, if requested, an ini file. It may be used in conjunction to previously loaded ldf files.

```

2812 \bbl@trace{Creating languages and reading ini files}
2813 \newcommand\babelprovide[2][{}]{%
2814   \let\bbl@savelangname\language
2815   \edef\bbl@savelocaleid{\the\localeid}%
2816   % Set name and locale id
2817   \edef\language{#2}%
2818   % \global\@namedef{\bbl@lcname@#2}{#2}%
2819   \bbl@id@assign
2820   \let\bbl@KVP@captions\@nil
2821   \let\bbl@KVP@import\@nil
2822   \let\bbl@KVP@main\@nil
2823   \let\bbl@KVP@script\@nil
2824   \let\bbl@KVP@language\@nil
2825   \let\bbl@KVP@hyphenrules\@nil % only for provide@new
2826   \let\bbl@KVP@mapfont\@nil
2827   \let\bbl@KVP@maparabic\@nil
2828   \let\bbl@KVP@mapdigits\@nil
2829   \let\bbl@KVP@intraspace\@nil
2830   \let\bbl@KVP@intrapenalty\@nil
2831   \let\bbl@KVP@onchar\@nil
2832   \let\bbl@KVP@alph\@nil
2833   \let\bbl@KVP@Alph\@nil
2834   \let\bbl@KVP@info\@nil % Ignored with import? Or error/warning?
2835   \bbl@forkv{#1}{% TODO - error handling
2836     \in@{/{}}{##1}%

```

```

2837 \ifin@
2838 \bbl@renewinikey##1\@{##2}%
2839 \else
2840 \bbl@csarg\def{KVP@##1}{##2}%
2841 \fi}%
2842 % == import, captions ==
2843 \ifx\bbl@KVP@import\@nil\else
2844 \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
2845 {\ifx\bbl@initoload\relax
2846 \begingroup
2847 \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2848 \InputIfFileExists{babel-#2.tex}{}}%
2849 \endgroup
2850 \else
2851 \xdef\bbl@KVP@import{\bbl@initoload}%
2852 \fi}%
2853 {}%
2854 \fi
2855 \ifx\bbl@KVP@captions\@nil
2856 \let\bbl@KVP@captions\bbl@KVP@import
2857 \fi
2858 % Load ini
2859 \bbl@ifunset{date#2}%
2860 {\bbl@provide@new{#2}}%
2861 {\bbl@ifblank{#1}%
2862 {\bbl@error
2863 {If you want to modify `#2' you must tell how in\\
2864 the optional argument. See the manual for the\\
2865 available options.}%
2866 {Use this macro as documented}}%
2867 {\bbl@provide@renew{#2}}}%
2868 % Post tasks
2869 \bbl@exp{\bbl@babelensure[exclude=\\today]{#2}}%
2870 \bbl@ifunset{\bbl@ensure@\\languagename}%
2871 {\bbl@exp{%
2872 \\DeclareRobustCommand\<bbl@ensure@\\languagename>[1]{%
2873 \\foreignlanguage{\\languagename}%
2874 {###1}}}%
2875 {}%
2876 \bbl@exp{%
2877 \\bbl@tglobal\<bbl@ensure@\\languagename>%
2878 \\bbl@tglobal\<bbl@ensure@\\languagename\space>%
2879 % At this point all parameters are defined if 'import'. Now we
2880 % execute some code depending on them. But what about if nothing was
2881 % imported? We just load the very basic parameters: ids and a few
2882 % more.
2883 \bbl@ifunset{\bbl@lname@#2}% TODO. Duplicated
2884 {\def\BabelBeforeIni##1##2{%
2885 \begingroup
2886 \catcode`\[=12 \catcode`\]=12 \catcode`\==12
2887 \catcode`\;=12 \catcode`\|=12 %
2888 \let\bbl@ini@captions@aux\@gobbles
2889 \def\bbl@inidate #####1.####2.####3.####4\relax #####5####6}%
2890 \bbl@read@ini{##1}{basic data}%
2891 \bbl@exportkey{chrng}{characters.ranges}{}%
2892 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2893 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2894 \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2895 \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%

```

```

2896         \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2897         \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2898         \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2899         \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2900         \bbl@exportkey{intsp}{typography.intraspace}{}%
2901         \ifx\bbl@initoload\relax\endinput\fi
2902     \endgroup}%
2903     \begingroup          % boxed, to avoid extra spaces:
2904     \ifx\bbl@initoload\relax
2905         \setbox\z@\hbox{\InputIfFileExists{babel-#2.tex}{}{}}%
2906     \else
2907         \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}{}}%
2908     \fi
2909     \endgroup}%
2910     {}%
2911     % == script, language ==
2912     % Override the values from ini or defines them
2913     \ifx\bbl@KVP@script\@nil\else
2914         \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2915     \fi
2916     \ifx\bbl@KVP@language\@nil\else
2917         \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2918     \fi
2919     % == onchar ==
2920     \ifx\bbl@KVP@onchar\@nil\else
2921         \bbl@luahyphenate
2922         \directlua{
2923             if Babel.locale_mapped == nil then
2924                 Babel.locale_mapped = true
2925                 Babel.linebreaking.add_before(Babel.locale_map)
2926                 Babel.loc_to_scr = {}
2927                 Babel.chr_to_loc = Babel.chr_to_loc or {}
2928             end}%
2929         \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2930     \ifin@
2931         \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
2932             \AddBabelHook{babel-onchar}{beforestart}{{\bbl@starthyphens}}%
2933         \fi
2934         \bbl@exp{\bbl@add\bbl@starthyphens
2935             {\bbl@patterns@lua{\language}}}%
2936         % TODO - error/warning if no script
2937         \directlua{
2938             if Babel.script_blocks['\bbl@cl{sbc}'] then
2939                 Babel.loc_to_scr[\the\localeid] =
2940                     Babel.script_blocks['\bbl@cl{sbc}']
2941                 Babel.locale_props[\the\localeid].lc = \the\localeid\space
2942                 Babel.locale_props[\the\localeid].lg = \the\@nameuse{1l\language}\space
2943             end
2944         }%
2945     \fi
2946     \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2947     \ifin@
2948         \bbl@ifunset{bbl@lsys\language}{\bbl@provide@lsys\language}{}%
2949         \bbl@ifunset{bbl@wdir\language}{\bbl@provide@dirs\language}{}%
2950         \directlua{
2951             if Babel.script_blocks['\bbl@cl{sbc}'] then
2952                 Babel.loc_to_scr[\the\localeid] =
2953                     Babel.script_blocks['\bbl@cl{sbc}']
2954             end}%

```

```

2955 \ifx\bb1@mapselect\@undefined
2956 \AtBeginDocument{%
2957 \expandafter\bb1@add\csname selectfont \endcsname{\bb1@mapselect}}%
2958 {\selectfont}}%
2959 \def\bb1@mapselect{%
2960 \let\bb1@mapselect\relax
2961 \edef\bb1@prefontid{\fontid\font}}%
2962 \def\bb1@mapdir##1{%
2963 {\def\language{##1}%
2964 \let\bb1@ifrestoring\@firstoftwo % To avoid font warning
2965 \bb1@switchfont
2966 \directlua{
2967 Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
2968 [\bb1@prefontid'] = \fontid\font\space}}}%
2969 \fi
2970 \bb1@exp{\bb1@add\bb1@mapselect{\bb1@mapdir{\language}}}%
2971 \fi
2972 % TODO - catch non-valid values
2973 \fi
2974 % == mapfont ==
2975 % For bidi texts, to switch the font based on direction
2976 \ifx\bb1@KVP@mapfont\@nil\else
2977 \bb1@ifsamestring{\bb1@KVP@mapfont}{direction}}%
2978 {\bb1@error{Option '\bb1@KVP@mapfont' unknown for\bb1@
2979 mapfont. Use 'direction'.}%
2980 {See the manual for details.}}}%
2981 \bb1@ifunset{\bb1@sys@\language}{\bb1@provide@sys{\language}}}%
2982 \bb1@ifunset{\bb1@wdir@\language}{\bb1@provide@dirs{\language}}}%
2983 \ifx\bb1@mapselect\@undefined
2984 \AtBeginDocument{%
2985 \expandafter\bb1@add\csname selectfont \endcsname{\bb1@mapselect}}%
2986 {\selectfont}}%
2987 \def\bb1@mapselect{%
2988 \let\bb1@mapselect\relax
2989 \edef\bb1@prefontid{\fontid\font}}%
2990 \def\bb1@mapdir##1{%
2991 {\def\language{##1}%
2992 \let\bb1@ifrestoring\@firstoftwo % avoid font warning
2993 \bb1@switchfont
2994 \directlua{Babel.fontmap
2995 [\the\csname bbl@wdir@##1\endcsname]%
2996 [\bb1@prefontid]=\fontid\font}}}%
2997 \fi
2998 \bb1@exp{\bb1@add\bb1@mapselect{\bb1@mapdir{\language}}}%
2999 \fi
3000 % == intraspace, intrapenalty ==
3001 % For CJK, East Asian, Southeast Asian, if interspace in ini
3002 \ifx\bb1@KVP@intraspace\@nil\else % We can override the ini or set
3003 \bb1@csarg\edef{intsp@#2}{\bb1@KVP@intraspace}%
3004 \fi
3005 \bb1@provide@intraspace
3006 % == hyphenate.other.locale ==
3007 \bb1@ifunset{\bb1@hyotl@\language}{}%
3008 {\bb1@csarg\bb1@replace{\hyotl@\language}{ }{ },}%
3009 \bb1@startcommands*\language}%
3010 \bb1@csarg\bb1@foreach{\hyotl@\language}{%
3011 \ifcase\bb1@engine
3012 \ifnum##1<257
3013 \SetHyphenMap{\BabelLower{##1}{##1}}%

```

```

3014         \fi
3015     \else
3016         \SetHyphenMap{\BabelLower{##1}{##1}}%
3017     \fi}%
3018 \bbl@endcommands}%
3019 % == hyphenate.other.script ==
3020 \bbl@ifunset{\bbl@hyots@language}{}%
3021 {\bbl@csarg\bbl@replace{hyots@language}{ }{,}}%
3022 \bbl@csarg\bbl@foreach{hyots@language}{%
3023     \ifcase\bbl@engine
3024     \ifnum##1<257
3025         \global\lccode##1=##1\relax
3026     \fi
3027     \else
3028         \global\lccode##1=##1\relax
3029     \fi}}%
3030 % == maparabic ==
3031 % Native digits, if provided in ini (TeX level, xe and lua)
3032 \ifcase\bbl@engine\else
3033     \bbl@ifunset{\bbl@dgnat@language}{}%
3034     {\expandafter\ifx\csname\bbl@dgnat@language\endcsname\@empty\else
3035         \expandafter\expandafter\expandafter
3036         \bbl@setdigits\csname\bbl@dgnat@language\endcsname
3037         \ifx\bbl@KVP@maparabic\@nil\else
3038             \ifx\bbl@latinarabic\@undefined
3039                 \expandafter\let\expandafter\@arabic
3040                 \csname\bbl@counter@language\endcsname
3041             \else % ie, if layout=counters, which redefines \@arabic
3042                 \expandafter\let\expandafter\bbl@latinarabic
3043                 \csname\bbl@counter@language\endcsname
3044             \fi
3045         \fi
3046     \fi}%
3047 \fi
3048 % == mapdigits ==
3049 % Native digits (lua level).
3050 \ifodd\bbl@engine
3051     \ifx\bbl@KVP@mapdigits\@nil\else
3052         \bbl@ifunset{\bbl@dgnat@language}{}%
3053         {\RequirePackage{luatexbase}%
3054         \bbl@activate@preotf
3055         \directlua{
3056             Babel = Babel or {} %%% -> presets in luababel
3057             Babel.digits_mapped = true
3058             Babel.digits = Babel.digits or {}
3059             Babel.digits[\the\localeid] =
3060                 table.pack(string.utfvalue('\bbl@c1{dgnat}'))
3061             if not Babel.numbers then
3062                 function Babel.numbers(head)
3063                     local LOCALE = luatexbase.registernumber'bbl@attr@locale'
3064                     local GLYPH = node.id'glyph'
3065                     local inmath = false
3066                     for item in node.traverse(head) do
3067                         if not inmath and item.id == GLYPH then
3068                             local temp = node.get_attribute(item, LOCALE)
3069                             if Babel.digits[temp] then
3070                                 local chr = item.char
3071                                 if chr > 47 and chr < 58 then
3072                                     item.char = Babel.digits[temp][chr-47]

```



```

3073             end
3074         end
3075         elseif item.id == node.id'math' then
3076             inmath = (item.subtype == 0)
3077         end
3078     end
3079     return head
3080 end
3081 end
3082 }}%
3083 \fi
3084 \fi
3085 % == alph, Alph ==
3086 % What if extras<lang> contains a \babel@save\@alph? It won't be
3087 % restored correctly when exiting the language, so we ignore
3088 % this change with the \bbl@alph@saved trick.
3089 \ifx\bbl@KVP@alph\@nil\else
3090     \toks@{\expandafter\expandafter\expandafter{%
3091         \csname extras\language\endcsname}%
3092         \bbl@exp{%
3093             \def\<extras\language>{%
3094                 \let\\bbl@alph@saved\\@alph
3095                 \the\toks@
3096                 \let\\@alph\\bbl@alph@saved
3097                 \\babel@save\\@alph
3098                 \let\\@alph<bbl@cntr@bbl@KVP@alph @\language>}}%
3099             \fi
3100             \ifx\bbl@KVP@Alph\@nil\else
3101                 \toks@{\expandafter\expandafter\expandafter{%
3102                     \csname extras\language\endcsname}%
3103                     \bbl@exp{%
3104                         \def\<extras\language>{%
3105                             \let\\bbl@Alph@saved\\@Alph
3106                             \the\toks@
3107                             \let\\@Alph\\bbl@Alph@saved
3108                             \\babel@save\\@Alph
3109                             \let\\@Alph<bbl@cntr@bbl@KVP@Alph @\language>}}%
3110                         \fi
3111                     % == require.babel in ini ==
3112                     % To load or reload the babel-*.tex, if require.babel in ini
3113                     \bbl@ifunset{bbl@rqtex@\language}{}%
3114                     {\expandafter\ifx\csname bbl@rqtex@\language\endcsname\@empty\else
3115                         \let\BabelBeforeIni\@gobbletwo
3116                         \chardef\atcatcode=\catcode\@
3117                         \catcode\@=11\relax
3118                         \InputIfFileExists{babel-\bbl@cs{rqtex@\language}.tex}{}}%
3119                         \catcode\@=\atcatcode
3120                         \let\atcatcode\relax
3121                     \fi}%
3122                     % == main ==
3123                     \ifx\bbl@KVP@main\@nil % Restore only if not 'main'
3124                         \let\language\bbl@savelangname
3125                         \chardef\localeid\bbl@savelocaleid\relax
3126                     \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 T<sub>E</sub>X. Non-digits characters are kept. The first macro is the generic “localized” command.

3127% TODO. Merge with \localenumeral:

[illegible]

```

3160 \def\bbl@provide@new#1{%
3161   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
3162   \@namedef{extras#1}{}%
3163   \@namedef{noextras#1}{}%
3164   \bbl@startcommands*{#1}{captions}%
3165   \ifx\bbl@KVP@captions\@nil % and also if import, implicit
3166     \def\bbl@tempb##1{% elt for \bbl@captionslist
3167       \ifx##1\@empty\else
3168         \bbl@exp{%
3169           \\SetString\\##1{%
3170             \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
3171           \expandafter\bbl@tempb
3172         \fi}%
3173     \expandafter\bbl@tempb\bbl@captionslist\@empty
3174   \else
3175     \ifx\bbl@initoload\relax
3176       \bbl@read@ini{\bbl@KVP@captions}{data}% Here letters cat = 11
3177     \else
3178       \bbl@read@ini{\bbl@initoload}{data}% Here all letters cat = 11
3179     \fi
3180     \bbl@after@ini
3181     \bbl@savestrings
3182   \fi
3183 \StartBabelCommands*{#1}{date}%
3184 \ifx\bbl@KVP@import\@nil

```

```

3185 \bbl@exp{%
3186 \\\SetString\\today{\\bbl@nocaption{today}{#1today}}}%
3187 \else
3188 \bbl@savetoday
3189 \bbl@savedate
3190 \fi
3191 \bbl@endcommands
3192 \bbl@ifunset{bbl@lname@#1}% TODO. Duplicated
3193 {\def\BabelBeforeIni##1##2{%
3194 \begingroup
3195 \catcode`\[=12 \catcode`\]=12 \catcode`\==12
3196 \catcode`\;=12 \catcode`\|=12 %
3197 \let\bbl@ini@captions@aux\@gobbletwo
3198 \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6{%
3199 \bbl@read@ini{##1}{basic data}%
3200 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3201 \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3202 \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3203 \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3204 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3205 \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3206 \bbl@exportkey{intsp}{typography.intraspace}{}%
3207 \bbl@exportkey{chrng}{characters.ranges}{}%
3208 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3209 \ifx\bbl@initoload\relax\endinput\fi
3210 \endgroup}%
3211 \begingroup % boxed, to avoid extra spaces:
3212 \ifx\bbl@initoload\relax
3213 \setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}}}%
3214 \else
3215 \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
3216 \fi
3217 \endgroup}%
3218 }%
3219 \bbl@exp{%
3220 \gdef\<#1hyphenmins>{%
3221 {\bbl@ifunset{bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
3222 {\bbl@ifunset{bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}%
3223 \bbl@provide@hyphens{#1}%
3224 \ifx\bbl@KVP@main\@nil\else
3225 \expandafter\main@language\expandafter{#1}%
3226 \fi}
3227 \def\bbl@provide@renew#1{%
3228 \ifx\bbl@KVP@captions\@nil\else
3229 \StartBabelCommands*{#1}{captions}%
3230 \bbl@read@ini{\bbl@KVP@captions}{data}% Here all letters cat = 11
3231 \bbl@after@ini
3232 \bbl@savestrings
3233 \EndBabelCommands
3234 \fi
3235 \ifx\bbl@KVP@import\@nil\else
3236 \StartBabelCommands*{#1}{date}%
3237 \bbl@savetoday
3238 \bbl@savedate
3239 \EndBabelCommands
3240 \fi
3241 % == hyphenrules ==
3242 \bbl@provide@hyphens{#1}}

```

The hyphenrules option is handled with an auxiliary macro.

```

3243 \def\bbbl@provide@hyphens#1{%
3244   \let\bbbl@tempa\relax
3245   \ifx\bbbl@KVP@hyphenrules\@nil\else
3246     \bbbl@replace\bbbl@KVP@hyphenrules{ }{,}%
3247     \bbbl@foreach\bbbl@KVP@hyphenrules{%
3248       \ifx\bbbl@tempa\relax    % if not yet found
3249         \bbbl@ifsamestring{##1}{+}%
3250         {\bbbl@exp{\addlanguage\<l@##1>}}}%
3251       {}%
3252       \bbbl@ifunset{l@##1}%
3253       {}%
3254       {\bbbl@exp{\let\bbbl@tempa\<l@##1>}}}%
3255     \fi}%
3256 \fi
3257 \ifx\bbbl@tempa\relax %      if no opt or no language in opt found
3258   \ifx\bbbl@KVP@import\@nil
3259     \ifx\bbbl@initoload\relax\else
3260       \bbbl@exp{%              and hyphenrules is not empty
3261         \bbbl@ifblank{\bbbl@cs{hyphr@#1}}%
3262         {}%
3263         {\let\bbbl@tempa\<l@\bbbl@cl{hyphr}>}}}%
3264     \fi
3265   \else % if importing
3266     \bbbl@exp{%              and hyphenrules is not empty
3267       \bbbl@ifblank{\bbbl@cs{hyphr@#1}}%
3268       {}%
3269       {\let\bbbl@tempa\<l@\bbbl@cl{hyphr}>}}}%
3270   \fi
3271 \fi
3272 \bbbl@ifunset{\bbbl@tempa}%      ie, relax or undefined
3273 {\bbbl@ifunset{l@#1}%          no hyphenrules found - fallback
3274   {\bbbl@exp{\adddialect\<l@#1>\language}}%
3275   {}}%
3276   {\bbbl@exp{\adddialect\<l@#1>\bbbl@tempa}}% found in opt list or ini
3277

```

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.

```

3278 \ifx\bbbl@readstream\@undefined
3279   \csname newread\endcsname\bbbl@readstream
3280 \fi
3281 \def\bbbl@inipreread#1=#2\@{%
3282   \bbbl@trim@def\bbbl@tempa{#1}% Redundant below !!
3283   \bbbl@trim\toks@{#2}%
3284   % Move trims here ??
3285   \bbbl@ifunset{\bbbl@KVP@\bbbl@section/\bbbl@tempa}%
3286   {\bbbl@exp{%
3287     \g@addto@macro\bbbl@inidata{%
3288       \bbbl@elt{\bbbl@section}{\bbbl@tempa}{\the\toks@}}}%
3289     \expandafter\bbbl@inireader\bbbl@tempa=#2\@}%
3290   {}}%
3291 \def\bbbl@read@ini#1#2{%
3292   \bbbl@csarg\edef\lini@{language}{#1}%
3293   \openin\bbbl@readstream=babel-#1.ini
3294   \ifeof\bbbl@readstream
3295     \bbbl@error
3296     {There is no ini file for the requested language\%

```

```

3297         (#1). Perhaps you misspelled it or your installation\\%
3298         is not complete.}%
3299         {Fix the name or reinstall babel.}%
3300     \else
3301         \bbl@exp{\def\\bbl@inidata{\\bbl@elt{identificacion}{tag.ini}{#1}}}%
3302         \let\bbl@section\empty
3303         \let\bbl@savestrings\empty
3304         \let\bbl@savetoday\empty
3305         \let\bbl@savestate\empty
3306         \let\bbl@inireader\bbl@iniskip
3307         \bbl@info{Importing #2 for \language\%
3308             from babel-#1.ini. Reported}%
3309         \loop
3310         \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3311             \endlinechar\m@ne
3312             \read\bbl@readstream to \bbl@line
3313             \endlinechar\^^M
3314             \ifx\bbl@line\empty\else
3315                 \expandafter\bbl@inline\bbl@line\bbl@inline
3316             \fi
3317         \repeat
3318         \bbl@foreach\bbl@renewlist{%
3319             \bbl@ifunset{\bbl@renew@##1}{\bbl@inisec[##1]\@}%
3320         \global\let\bbl@renewlist\empty
3321         % Ends last section. See \bbl@inisec
3322         \def\bbl@elt##1##2{\bbl@inireader##1=##2\@}%
3323         \bbl@cs{renew@\bbl@section}%
3324         \global\bbl@csarg\let{renew@\bbl@section}\relax
3325         \bbl@cs{secpost@\bbl@section}%
3326         \bbl@csarg{\global\expandafter\let}{inidata@\language}\bbl@inidata
3327         \bbl@exp{\\bbl@add@list\\bbl@ini@loaded{\language}}%
3328         \bbl@tglobal\bbl@ini@loaded
3329     \fi}
3330 \def\bbl@inline#1\bbl@inline{%
3331     \@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.

```

3332 \def\bbl@iniskip#1\@{%           if starts with ;
3333 \def\bbl@inisec[#1]#2\@{%       if starts with opening bracket
3334     \def\bbl@elt##1##2{%
3335         \expandafter\toks@\expandafter{%
3336             \expandafter{\bbl@section}{##1}{##2}}%
3337         \bbl@exp{%
3338             \\g@addto@macro\\bbl@inidata{\\bbl@elt\the\toks@}%
3339             \bbl@inireader##1=##2\@}%
3340         \bbl@cs{renew@\bbl@section}%
3341         \global\bbl@csarg\let{renew@\bbl@section}\relax
3342         \bbl@cs{secpost@\bbl@section}%
3343         % The previous code belongs to the previous section.
3344         % -----
3345         % Now start the current one.
3346         \in@{=date.}{#1}%
3347         \ifin@
3348             \lowercase{\def\bbl@tempa{#1=}}%
3349             \bbl@replace\bbl@tempa{=date.gregorian}{}%

```

```

3350 \bbl@replace\bbl@tempa{=date.}{}%
3351 \bbl@replace\bbl@tempa{=}{}%
3352 \in@{.licr.}{#1.}%
3353 \ifin@
3354 \ifcase\bbl@engine
3355 \bbl@replace\bbl@tempa{.licr}{}%
3356 \else
3357 \let\bbl@tempa\relax
3358 \fi
3359 \fi
3360 \ifx\bbl@tempa\relax\else
3361 \bbl@exp{%
3362 \def\bbl@inikv@#1>####1=####2\\@{%
3363 \\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
3364 \fi
3365 \fi
3366 \def\bbl@section{#1}%
3367 \def\bbl@elt##1##2{%
3368 \@namedef\bbl@KVP@#1/##1}{}%
3369 \bbl@cs{renew@#1}%
3370 \bbl@cs{secpre@#1}% pre-section `hook'
3371 \bbl@ifunset\bbl@inikv@#1}%
3372 {\let\bbl@inireader\bbl@iniskip}%
3373 {\bbl@exp{\let\\bbl@inireader<\bbl@inikv@#1>}}
3374 \let\bbl@renewlist\@empty
3375 \def\bbl@renewinikey#1/#2\@#3{%
3376 \bbl@ifunset\bbl@renew@#1}%
3377 {\bbl@add@list\bbl@renewlist{#1}}%
3378 {}%
3379 \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>.

```

3380 \def\bbl@inikv#1=#2\@{%      key=value
3381 \bbl@trim\def\bbl@tempa{#1}%
3382 \bbl@trim\toks@{#2}%
3383 \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.

```

3384 \def\bbl@exportkey#1#2#3{%
3385 \bbl@ifunset\bbl@kv@#2}%
3386 {\bbl@csarg\gdef{#1@\languagename}{#3}}%
3387 {\expandafter\ifx\csname\bbl@kv@#2\endcsname\@empty
3388 \bbl@csarg\gdef{#1@\languagename}{#3}}%
3389 \else
3390 \bbl@exp{\global\let<\bbl@#1@\languagename><\bbl@kv@#2>}%
3391 \fi}}

```

Key-value pairs are treated differently depending on the section in the ini file. The following macros are the readers for identification and typography. Note \bbl@secpost@identification is called always (via \bbl@inisec), while \bbl@after@ini must be called explicitly after \bbl@read@ini if necessary.

```

3392 \def\bbl@iniwarning#1{%
3393 \bbl@ifunset\bbl@kv@identification.warning#1}{}%
3394 {\bbl@warning{%
3395 From babel-\bbl@cs{lini@\languagename}.ini:\\
3396 \bbl@cs{kv@identification.warning#1}\\
3397 Reported }}}
3398 \let\bbl@inikv@identification\bbl@inikv

```

```

3399 \def\bbl@secpost@identification{%
3400   \bbl@iniwarning{}%
3401   \ifcase\bbl@engine
3402     \bbl@iniwarning{.pdflatex}%
3403   \or
3404     \bbl@iniwarning{.lualatex}%
3405   \or
3406     \bbl@iniwarning{.xelatex}%
3407   \fi%
3408   \bbl@exportkey{elname}{identification.name.english}{}%
3409   \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
3410     {\csname bbl@elname@language\endcsname}}%
3411   \bbl@exportkey{lbcpr}{identification.tag.bcp47}{}%
3412   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3413   \bbl@exportkey{esname}{identification.script.name}{}%
3414   \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
3415     {\csname bbl@esname@language\endcsname}}%
3416   \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
3417   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}}
3418 \let\bbl@inikv@typography\bbl@inikv
3419 \let\bbl@inikv@characters\bbl@inikv
3420 \let\bbl@inikv@numbers\bbl@inikv
3421 \def\bbl@inikv@counters#1=#2\@{%
3422   \bbl@ifsamestring{#1}{digits}%
3423     {\bbl@error{The counter name 'digits' is reserved for mapping\\
3424       decimal digits}%
3425       {Use another name.}}%
3426   {}%
3427   \def\bbl@tempc{#1}%
3428   \bbl@trim\def\bbl@tempb*}{#2}%
3429   \in@{.1$}{#1$}%
3430   \ifin@
3431     \bbl@replace\bbl@tempc{.1}{}%
3432     \bbl@csarg\protected@xdef{cntr@#1\bbl@tempc @language}{%
3433       \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3434   \fi
3435   \in@{.F.}{#1}%
3436   \ifin@else\in@{.S.}{#1}\fi
3437   \ifin@
3438     \bbl@csarg\protected@xdef{cntr@#1@language}{\bbl@tempb*}%
3439   \else
3440     \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3441     \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \ \
3442     \bbl@csarg{\global\expandafter\let}{cntr@#1@language}\bbl@tempa
3443   \fi}
3444 \def\bbl@after@ini{%
3445   \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
3446   \bbl@exportkey{rgtm}{typography.righthyphenmin}{3}%
3447   \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3448   \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3449   \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3450   \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3451   \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3452   \bbl@exportkey{intsp}{typography.intraspace}{}%
3453   \bbl@exportkey{jstfy}{typography.justify}{w}%
3454   \bbl@exportkey{chrng}{characters.ranges}{}%
3455   \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3456   \bbl@exportkey{rqtex}{identification.require.babel}{}%
3457   \bbl@toglobal\bbl@savetoday

```

```
3458 \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.

```
3459 \ifcase\bbl@engine
3460 \bbl@csarg\def{inikv@captions.licr}#1=#2\@@{%
3461 \bbl@ini@captions@aux{#1}{#2}}
3462 \else
3463 \def\bbl@inikv@captions#1=#2\@@{%
3464 \bbl@ini@captions@aux{#1}{#2}}
3465 \fi
```

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

```
3466 \def\bbl@ini@captions@aux#1#2{%
3467 \bbl@trim\def\bbl@tempa{#1}%
3468 \bbl@ifblank{#2}%
3469 {\bbl@exp{%
3470 \toks{\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
3471 {\bbl@trim\toks{#2}}%
3472 \bbl@exp{%
3473 \bbl@add\bbl@savestrings{%
3474 \SetString\<\bbl@tempa name>{\the\toks}}}}
```

TODO. Document

```
3475 % Arguments are _not_ protected.
3476 \let\bbl@calendar\@empty
3477 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3478 \def\bbl@localedate#1#2#3#4{%
3479 \begingroup
3480 \ifx\@empty#1\@empty\else
3481 \let\bbl@ld@calendar\@empty
3482 \let\bbl@ld@format\@empty
3483 \edef\bbl@tempa{\zap@space#1 \@empty}%
3484 \def\bbl@tempb##1=##2\@@{\@namedef{\bbl@ld##1}{##2}}%
3485 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
3486 \edef\bbl@tempc{\bbl@ld@calendar\bbl@ld@format}%
3487 \ifx\bbl@tempc\@empty\else
3488 \edef\bbl@calendar{%
3489 \ifx\bbl@ld@calendar\@empty
3490 %gregorian%
3491 \else
3492 \bbl@ld@calendar
3493 \fi
3494 \ifx\bbl@ld@format\@empty\else
3495 .\bbl@ld@format
3496 \fi}%
3497 \fi
3498 \def\bbl@tempb{gregorian}%
3499 \ifx\bbl@calendar\bbl@tempb
3500 \let\bbl@calendar\@empty
3501 \fi
3502 \fi
3503 \@nameuse{\bbl@date@\language @\bbl@calendar}{#2}{#3}{#4}%
3504 \endgroup}
3505 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3506 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3507 \bbl@trim\def\bbl@tempa{#1.#2}%
3508 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savestate
```



```

3509 {\bbl@trim@def\bbl@tempa{#3}%
3510 \bbl@trim\toks@{#5}%
3511 \@temptokena\expandafter{\bbl@savestate}%
3512 \bbl@exp{% Reverse order - in ini last wins
3513 \def\\bbl@savestate{%
3514 \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3515 \the\@temptokena}}}%
3516 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3517 {\bbl@ifblank{#6}%
3518 {\let\bbl@tempb\@empty}%
3519 {\lowercase{\def\bbl@tempb{#6}}}%
3520 \bbl@trim@def\bbl@toreplace{#5}%
3521 \bbl@TG@date
3522 \bbl@ifunset{\bbl@date@\language @}%
3523 {\global\bbl@csarg\let{date@\language @}\bbl@toreplace
3524 % TODO. Move to a better place.
3525 \bbl@exp{%
3526 \gdef\<\language date>{\protect\<\language date >}%
3527 \gdef\<\language date >####1####2####3{%
3528 \\bbl@usedategroupttrue
3529 \<bbl@ensure@\language >%
3530 \\localedate{####1}{####2}{####3}}}%
3531 \\bbl@add\\bbl@savetoday{%
3532 \\SetString\\today{%
3533 \<\language date>%
3534 {\the\year}{\the\month}{\the\day}}}%
3535 {}%
3536 \ifx\bbl@tempb\@empty\else
3537 \global\bbl@csarg\let{date@\language @}\bbl@tempb\bbl@toreplace
3538 \fi}%
3539 {}%

```

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.

```

3540 \let\bbl@calendar\@empty
3541 \newcommand\BabelDateSpace{\nobreakspace}
3542 \newcommand\BabelDateDot{.\@}
3543 \newcommand\BabelDated[1]{\number#1}
3544 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3545 \newcommand\BabelDateM[1]{\number#1}
3546 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3547 \newcommand\BabelDateMMMM[1]{%
3548 \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3549 \newcommand\BabelDatey[1]{\number#1}%
3550 \newcommand\BabelDateyy[1]{%
3551 \ifnum#1<10 0\number#1 %
3552 \else\ifnum#1<100 \number#1 %
3553 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3554 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3555 \else
3556 \bbl@error
3557 {Currently two-digit years are restricted to the\
3558 range 0-9999.}%
3559 {There is little you can do. Sorry.}%
3560 \fi\fi\fi\fi}
3561 \newcommand\BabelDateyyyy[1]{\number#1} % FIXME - add leading 0
3562 \def\bbl@replace@finish@iii#1{%
3563 \bbl@exp{\def\\#1####1####2####3{\the\toks@}}

```

```

3564 \def\bbl@TG@date{%
3565   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3566   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3567   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3568   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3569   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3570   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3571   \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3572   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3573   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3574   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3575   \bbl@replace\bbl@toreplace{[y|]}{\bbl@datecctr[####1|]}%
3576   \bbl@replace\bbl@toreplace{[m|]}{\bbl@datecctr[####2|]}%
3577   \bbl@replace\bbl@toreplace{[d|]}{\bbl@datecctr[####3|]}%
3578 % Note after \bbl@replace \toks@ contains the resulting string.
3579 % TODO - Using this implicit behavior doesn't seem a good idea.
3580   \bbl@replace@finish@iii\bbl@toreplace}
3581 \def\bbl@datecctr[#1|#2]{\localenumeral{#2}{#1}}

```

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

```

3582 \def\bbl@provide@lsys#1{%
3583   \bbl@ifunset{bbl@lname@#1}%
3584     {\bbl@ini@basic{#1}}%
3585   }%
3586   \bbl@csarg\let{lsys@#1}\@empty
3587   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3588   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3589   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3590   \bbl@ifunset{bbl@lname@#1}{%
3591     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3592   \ifcase\bbl@engine\or\or
3593     \bbl@ifunset{bbl@prehc@#1}{%
3594       {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3595       }%
3596       {\bbl@csarg\bbl@add@list{lsys@#1}{HyphenChar="200B}}}%
3597   \fi
3598   \bbl@csarg\bbl@toglobal{lsys@#1}}

```

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

```

3599 \def\bbl@ini@basic#1{%
3600   \def\BabelBeforeIni##1##2{%
3601     \begingroup
3602       \bbl@add\bbl@secpost@identification{\closein\bbl@readstream}%
3603       \catcode`\[=12 \catcode`\]=12 \catcode`\==12
3604       \catcode`\;=12 \catcode`\|=12 %
3605       \bbl@read@ini{##1}{font and identification data}%
3606       \endinput % babel- .tex may contain onlypreamble's
3607     \endgroup}% boxed, to avoid extra spaces:
3608   {\setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}}}%

```

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

```

3609 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={
3610   \ifx\#1% % \ before, in case #1 is multiletter
3611     \bbl@exp{%

```

```

3612 \def\\bbl@tempa####1{%
3613 \ifcase>####1\space\the\toks@<else>\\@ctrerr<fi>}}%
3614 \else
3615 \toks@\\expandafter{\the\toks@\\or #1}%
3616 \expandafter\bbl@buildifcase
3617 \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).

```

3618 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@\\language}{#2}}
3619 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3620 \newcommand\localecounter[2]{%
3621 \expandafter\bbl@localecntr
3622 \expandafter{\number\csname c@#2\endcsname}{#1}}
3623 \def\bbl@alphnumeral#1#2{%
3624 \expandafter\bbl@alphnumeral@i\number#2 76543210\\@@{#1}}
3625 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\\@@#9{%
3626 \ifcase\\car#8\\nil\\or % Currently <10000, but prepared for bigger
3627 \bbl@alphnumeral@ii{#9}000000#1\\or
3628 \bbl@alphnumeral@ii{#9}00000#1#2\\or
3629 \bbl@alphnumeral@ii{#9}0000#1#2#3\\or
3630 \bbl@alphnumeral@ii{#9}000#1#2#3#4\\else
3631 \bbl@alphnum@invalid{>9999}%
3632 \fi}
3633 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3634 \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\\language}%
3635 {\bbl@cs{cntr@#1.4@\\language}{#5}
3636 \bbl@cs{cntr@#1.3@\\language}{#6}
3637 \bbl@cs{cntr@#1.2@\\language}{#7}
3638 \bbl@cs{cntr@#1.1@\\language}{#8}
3639 \ifnum#6#7#8>\\z@ % TODO. An ad hoc rule for Greek. Ugly.
3640 \bbl@ifunset{bbl@cntr@#1.S.321@\\language}{}}%
3641 {\bbl@cs{cntr@#1.S.321@\\language}}%
3642 \fi}%
3643 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\\language}}
3644 \def\bbl@alphnum@invalid#1{%
3645 \bbl@error{Alphabetic numeral too large (#1)}%
3646 {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.

```

3647 \newcommand\localeinfo[1]{%
3648 \bbl@ifunset{bbl@\\csname bbl@info@#1\endcsname @\\language}%
3649 {\bbl@error{I've found no info for the current locale.\\%
3650 The corresponding ini file has not been loaded\\%
3651 Perhaps it doesn't exist}%
3652 {See the manual for details.}}%
3653 {\bbl@cs{\\csname bbl@info@#1\endcsname @\\language}}
3654 % \@namedef{bbl@info@name.locale}{lcname}
3655 \@namedef{bbl@info@tag.ini}{lini}
3656 \@namedef{bbl@info@name.english}{elname}
3657 \@namedef{bbl@info@name.opentype}{lname}
3658 \@namedef{bbl@info@tag.bcp47}{lbc}
3659 \@namedef{bbl@info@tag.opentype}{lotf}

```

```

3660 \@namedef{bbl@info@script.name}{esname}
3661 \@namedef{bbl@info@script.name.opentype}{sname}
3662 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3663 \@namedef{bbl@info@script.tag.opentype}{sotf}
3664 \let\bbl@ensureinfo\@gobble
3665 \newcommand\BabelEnsureInfo{%
3666   \def\bbl@ensureinfo##1{%
3667     \ifx\InputIfFileExists\undefined\else % not in plain
3668       \bbl@ifunset{bbl@lname@##1}{\bbl@ini@basic{##1}}{}%
3669     \fi}}

```

More general, but non-expandable, is `\getlocaleproperty`. To inspect every possible loaded ini, we define `\LocaleForEach`, where `\bbl@ini@loaded` is a comma-separated list of locales, built by `\bbl@read@ini`.

```

3670 \newcommand\getlocaleproperty[3]{%
3671   \let#1\relax
3672   \def\bbl@elt##1##2##3{%
3673     \bbl@ifsamestring{##1/##2}{##3}%
3674     {\providecommand#1{##3}%
3675     \def\bbl@elt####1####2####3{}}}%
3676   {}}%
3677   \bbl@cs{inidata@#2}%
3678   \ifx#1\relax
3679     \bbl@error
3680     {Unknown key for locale '#2':\%
3681     #3\%
3682     \string#1 will be set to \relax}%
3683   {Perhaps you misspelled it.}%
3684   \fi}
3685 \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.

```

3686 \newcommand\babeladjust[1]{% TODO. Error handling.
3687   \bbl@forkv{#1}{%
3688     \bbl@ifunset{bbl@ADJ@##1@##2}%
3689     {\bbl@cs{ADJ@##1}{##2}}%
3690     {\bbl@cs{ADJ@##1@##2}}}
3691 %
3692 \def\bbl@adjust@lua#1#2{%
3693   \ifvmode
3694     \ifnum\currentgrouplevel=\z@
3695       \directlua{ Babel.#2 }%
3696       \expandafter\expandafter\expandafter\@gobble
3697     \fi
3698   \fi
3699   {\bbl@error % The error is gobbled if everything went ok.
3700     {Currently, #1 related features can be adjusted only\%
3701     in the main vertical list.}%
3702     {Maybe things change in the future, but this is what it is.}}}
3703 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3704   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3705 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3706   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3707 \@namedef{bbl@ADJ@bidi.text@on}{%
3708   \bbl@adjust@lua{bidi}{bidi_enabled=true}}

```

```

3709 \@namedef{bbl@ADJ@bidi.text@off}{%
3710   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3711 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3712   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3713 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3714   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3715 %
3716 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3717   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3718 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3719   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3720 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3721   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3722 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3723   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3724 %
3725 \def\bbl@adjust@layout#1{%
3726   \ifvmode
3727     #1%
3728     \expandafter\@gobble
3729   \fi
3730   {\bbl@error   % The error is gobbled if everything went ok.
3731     {Currently, layout related features can be adjusted only\\%
3732       in vertical mode.}%
3733     {Maybe things change in the future, but this is what it is.}}}
3734 \@namedef{bbl@ADJ@layout.tabular@on}{%
3735   \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
3736 \@namedef{bbl@ADJ@layout.tabular@off}{%
3737   \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
3738 \@namedef{bbl@ADJ@layout.lists@on}{%
3739   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3740 \@namedef{bbl@ADJ@layout.lists@off}{%
3741   \bbl@adjust@layout{\let\list\bbl@OL@list}}
3742 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
3743   \bbl@activateposthyphen}
3744 %
3745 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3746   \bbl@bcpallowedtrue}
3747 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3748   \bbl@bcpallowedfalse}
3749 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3750   \def\bbl@bcp@prefix{#1}}
3751 \def\bbl@bcp@prefix{bcp47-}
3752 \@namedef{bbl@ADJ@autoload.options}#1{%
3753   \def\bbl@autoload@options{#1}}
3754 \let\bbl@autoload@bcptoptions\@empty
3755 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3756   \def\bbl@autoload@bcptoptions{#1}}
3757 % TODO: use babel name, override
3758 %
3759 % As the final task, load the code for lua.
3760 %
3761 \ifx\directlua\@undefined\else
3762   \ifx\bbl@luapatterns\@undefined
3763     \input luababel.def
3764   \fi
3765 \fi
3766 </core>

```

A proxy file for switch.def

```
3767 <*kernel>
3768 \let\bbl@onlyswitch\@empty
3769 \input babel.def
3770 \let\bbl@onlyswitch\@undefined
3771 </kernel>
3772 <*patterns>
```

## 11 Loading hyphenation patterns

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

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

```
3773 <<Make sure ProvidesFile is defined>>
3774 \ProvidesFile{hyphen.cfg}[\<date>\<version>] Babel hyphens]
3775 \xdef\bbl@format{\jobname}
3776 \def\bbl@version{\<version>}
3777 \def\bbl@date{\<date>}
3778 \ifx\AtBeginDocument\@undefined
3779   \def\@empty{}
3780   \let\orig@dump\dump
3781   \def\dump{%
3782     \ifx\@ztryfc\@undefined
3783     \else
3784       \toks0=\expandafter{\@preamblecmds}%
3785       \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
3786       \def\@begindocumenthook{}%
3787     \fi
3788     \let\dump\orig@dump\let\orig@dump\@undefined\dump}
3789 \fi
3790 <<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.

```
3791 \def\process@line#1#2 #3 #4 {%
3792   \ifx=#1%
3793     \process@synonym{#2}%
3794   \else
3795     \process@language{#1#2}{#3}{#4}%
3796   \fi
3797   \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.

```
3798 \toks@{}
3799 \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.

```

3800 \def\process@synonym#1{%
3801   \ifnum\last@language=\m@ne
3802     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
3803   \else
3804     \expandafter\chardef\csname l@#1\endcsname\last@language
3805     \wlog{\string\l@#1=\string\language\the\last@language}%
3806     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
3807       \csname\language\hyphenmins\endcsname
3808     \let\bbl@elt\relax
3809     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
3810   \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 `\langhyphenmins` macro. When no assignments were made we provide a default setting.

Some pattern files contain changes to the `\lccode` and `\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.

```

3811 \def\process@language#1#2#3{%
3812   \expandafter\addlanguage\csname l@#1\endcsname
3813   \expandafter\language\csname l@#1\endcsname
3814   \edef\language{#1}%
3815   \bbl@hook@everylanguage{#1}%
3816   % > luatex
3817   \bbl@get@enc#1::\@@@

```

```

3818 \begingroup
3819 \lefthyphenmin\m@ne
3820 \bbl@hook@loadpatterns{#2}%
3821 % > luatex
3822 \ifnum\lefthyphenmin=\m@ne
3823 \else
3824 \expandafter\xdef\csname #1hyphenmins\endcsname{%
3825 \the\lefthyphenmin\the\righthyphenmin}%
3826 \fi
3827 \endgroup
3828 \def\bbl@tempa{#3}%
3829 \ifx\bbl@tempa\@empty\else
3830 \bbl@hook@loadexceptions{#3}%
3831 % > luatex
3832 \fi
3833 \let\bbl@elt\relax
3834 \def\bbl@languages{%
3835 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
3836 \ifnum\the\language=\z@
3837 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
3838 \set@hyphenmins\tw@\thr@\relax
3839 \else
3840 \expandafter\expandafter\expandafter\set@hyphenmins
3841 \csname #1hyphenmins\endcsname
3842 \fi
3843 \the\toks@
3844 \toks@{}%
3845 \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.

```

3846 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}

```

Now, hooks are defined. For efficiency reasons, they are dealt here in a special way. Besides luatex, format-specific configuration files are taken into account. loadkernel currently loads nothing, but define some basic macros instead.

```

3847 \def\bbl@hook@everylanguage#1{}
3848 \def\bbl@hook@loadpatterns#1{\input #1\relax}
3849 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
3850 \def\bbl@hook@loadkernel#1{%
3851 \def\addlanguage{\csname newlanguage\endcsname}%
3852 \def\adddialect##1##2{%
3853 \global\chardef##1##2\relax
3854 \wlog{\string##1 = a dialect from \string\language##2}}%
3855 \def\iflanguage##1{%
3856 \expandafter\ifx\csname l@##1\endcsname\relax
3857 \nolannerr{##1}%
3858 \else
3859 \ifnum\csname l@##1\endcsname=\language
3860 \expandafter\expandafter\expandafter\@firstoftwo
3861 \else
3862 \expandafter\expandafter\expandafter\@secondoftwo
3863 \fi
3864 \fi}%
3865 \def\providehyphenmins##1##2{%
3866 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
3867 \namedef{##1hyphenmins}{##2}%
3868 \fi}%

```



```

3869 \def\set@hyphenmins##1##2{%
3870   \lefthyphenmin##1\relax
3871   \righthyphenmin##2\relax}%
3872 \def\selectlanguage{%
3873   \errhelp{Selecting a language requires a package supporting it}%
3874   \errmessage{Not loaded}}%
3875 \let\foreignlanguage\selectlanguage
3876 \let\otherlanguage\selectlanguage
3877 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
3878 \def\bbl@usehooks##1##2{% TODO. Temporary!!
3879 \def\setlocale{%
3880   \errhelp{Find an armchair, sit down and wait}%
3881   \errmessage{Not yet available}}%
3882 \let\uselocale\setlocale
3883 \let\locale\setlocale
3884 \let\selectlocale\setlocale
3885 \let\localename\setlocale
3886 \let\textlocale\setlocale
3887 \let\textlanguage\setlocale
3888 \let\languagetext\setlocale}
3889 \begingroup
3890 \def\AddBabelHook#1#2{%
3891   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
3892     \def\next{\toks1}%
3893     \else
3894       \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
3895     \fi
3896     \next}
3897 \ifx\directlua\@undefined
3898   \ifx\XeTeXinputencoding\@undefined\else
3899     \input xebabel.def
3900   \fi
3901 \else
3902   \input luababel.def
3903 \fi
3904 \openin1 = babel-\bbl@format.cfg
3905 \ifeof1
3906 \else
3907   \input babel-\bbl@format.cfg\relax
3908 \fi
3909 \closein1
3910 \endgroup
3911 \bbl@hook@loadkernel{switch.def}

```

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

```

3912 \openin1 = language.dat

```

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

```

3913 \def\languagename{english}%
3914 \ifeof1
3915   \message{I couldn't find the file language.dat,\space
3916           I will try the file hyphen.tex}
3917   \input hyphen.tex\relax
3918   \chardef\l@english\z@
3919 \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`.

```
3920 \last@language\m@ne
```

We now read lines from the file until the end is found. While reading from the input, it is useful to switch off recognition of the end-of-line character. This saves us stripping off spaces from the contents of the control sequence.

```
3921 \loop
3922 \endlinechar\m@ne
3923 \read1 to \bbl@line
3924 \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.

```
3925 \if T\ifeof1F\fi T\relax
3926 \ifx\bbl@line\@empty\else
3927 \edef\bbl@line{\bbl@line\space\space\space}%
3928 \expandafter\process@line\bbl@line\relax
3929 \fi
3930 \repeat
```

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

```
3931 \begingroup
3932 \def\bbl@elt#1#2#3#4{%
3933 \global\language=#2\relax
3934 \gdef\language{#1}%
3935 \def\bbl@elt##1##2##3##4{}}%
3936 \bbl@languages
3937 \endgroup
3938 \fi
3939 \closein1
```

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

```
3940 \if/\the\toks@\else
3941 \errhelp{language.dat loads no language, only synonyms}
3942 \errmessage{Orphan language synonym}
3943 \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.

```
3944 \let\bbl@line\@undefined
3945 \let\process@line\@undefined
3946 \let\process@synonym\@undefined
3947 \let\process@language\@undefined
3948 \let\bbl@get@enc\@undefined
3949 \let\bbl@hyph@enc\@undefined
3950 \let\bbl@tempa\@undefined
3951 \let\bbl@hook@loadkernel\@undefined
3952 \let\bbl@hook@everylanguage\@undefined
3953 \let\bbl@hook@loadpatterns\@undefined
3954 \let\bbl@hook@loadexceptions\@undefined
3955 </patterns>
```

Here the code for `iniTEX` ends.

## 12 Font handling with fontspec

Add the bidi handler just before luaotfload, which is loaded by default by LaTeX. Just in case, consider the possibility it has not been loaded. First, a couple of definitions related to bidi [misplaced].

```
3956 <<(*More package options)>> ≡
3957 \chardef\bbl@bidimode\z@
3958 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
3959 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
3960 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
3961 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
3962 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
3963 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
3964 <</More package options>>
```

With explicit languages, we could define the font at once, but we don't. Just wait and see if the language is actually activated. `bbl@font` replaces hardcoded font names inside `\..family` by the corresponding macro `\..default`.

```
3965 <<(*Font selection)>> ≡
3966 \bbl@trace{Font handling with fontspec}
3967 \@onlypreamble\babelfont
3968 \newcommand\babelfont[2][% 1=langs/scripts 2=fam
3969   \bbl@foreach{#1}{%
3970     \expandafter\ifx\csname date##1\endcsname\relax
3971       \IfFileExists{babel-##1.tex}%
3972         {\babelprovide{##1}}%
3973         {}%
3974     \fi}%
3975 \edef\bbl@tempa{#1}%
3976 \def\bbl@tempb{#2}% Used by \bbl@bblfont
3977 \ifx\fontspec\undefined
3978   \usepackage{fontspec}%
3979 \fi
3980 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
3981 \bbl@bblfont}
3982 \newcommand\bbl@bblfont[2][% 1=features 2=fontname, @font=rm|sf|tt
3983   \bbl@ifunset{\bbl@tempb family}%
3984   {\bbl@providedefam{\bbl@tempb}}%
3985   {\bbl@exp{%
3986     \\\bbl@sreplace\<\bbl@tempb family >%
3987     {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
3988 % For the default font, just in case:
3989 \bbl@ifunset{\bbl@lsys\@languagenome}{\bbl@provide@lsys{\@languagenome}}}%
3990 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
3991   {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
3992   \bbl@exp{%
3993     \let\<\bbl@\bbl@tempb dflt@\@languagenome>\<\bbl@\bbl@tempb dflt@>%
3994     \\\bbl@font@set\<\bbl@\bbl@tempb dflt@\@languagenome>%
3995     \<\bbl@tempb default>\<\bbl@tempb family>}}}%
3996   {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
3997     \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:

```
3998 \def\bbl@providedefam#1{%
3999   \bbl@exp{%
4000     \\\newcommand\<#1default>{}% Just define it
4001     \\\bbl@add@list\<\bbl@font@fams{#1}%
4002     \\\DeclareRobustCommand\<#1family>{%

```

```

4003      \\\not@math@alphabet\<#1family>\relax
4004      \\\fontfamily\<#1default>\\selectfont}%
4005      \\\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.

```

4006 \def\bbl@nostdfont#1{%
4007   \bbl@ifunset{bbl@WFF@f@family}%
4008   {\bbl@csarg\gdef{WFF@f@family}}}% Flag, to avoid dupl warns
4009   \bbl@infowarn{The current font is not a babel standard family:\\%
4010     #1%
4011     \fontname\font\\%
4012     There is nothing intrinsically wrong with this warning, and\\%
4013     you can ignore it altogether if you do not need these\\%
4014     families. But if they are used in the document, you should be\\%
4015     aware 'babel' will no set Script and Language for them, so\\%
4016     you may consider defining a new family with \string\babelfont.\\%
4017     See the manual for further details about \string\babelfont.\\%
4018     Reported}}
4019   {}}%
4020 \gdef\bbl@switchfont{%
4021   \bbl@ifunset{bbl@lsys@\languageName}{\bbl@provide@lsys{\languageName}}}%
4022   \bbl@exp{% eg Arabic -> arabic
4023     \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}}%
4024   \bbl@foreach\bbl@font@fams{%
4025     \bbl@ifunset{bbl@##1dflt@\languageName}% (1) language?
4026     {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}% (2) from script?
4027       {\bbl@ifunset{bbl@##1dflt@}% 2=F - (3) from generic?
4028         {}% 123=F - nothing!
4029         {\bbl@exp{% 3=T - from generic
4030           \global\let\<bbl@##1dflt@\languageName>%
4031             \<bbl@##1dflt@>}}}%
4032         {\bbl@exp{% 2=T - from script
4033           \global\let\<bbl@##1dflt@\languageName>%
4034             \<bbl@##1dflt@*\bbl@tempa>}}}%
4035         {}}% 1=T - language, already defined
4036   \def\bbl@tempa{\bbl@nostdfont}}}%
4037   \bbl@foreach\bbl@font@fams{% don't gather with prev for
4038     \bbl@ifunset{bbl@##1dflt@\languageName}%
4039     {\bbl@cs{famrst@##1}%
4040     \global\bbl@csarg\let{famrst@##1}\relax}%
4041     {\bbl@exp{% order is relevant
4042       \\\bbl@add\\originalTeX{%
4043         \\\bbl@font@rst{\bbl@cl{##1dflt}}}%
4044         \<##1default>\<##1family>{##1}}}%
4045       \\\bbl@font@set\<bbl@##1dflt@\languageName>% the main part!
4046       \<##1default>\<##1family>}}}%
4047   \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.

```

4048 \ifx\fontfamily\undefined\else % if latex
4049   \ifcase\bbl@engine % if pdftex
4050     \let\bbl@ckeckstdfonts\relax
4051   \else
4052     \def\bbl@ckeckstdfonts{%
4053       \begingroup
4054       \global\let\bbl@ckeckstdfonts\relax
4055       \let\bbl@tempa\empty

```

```

4056 \bbl@foreach\bbl@font@fams{%
4057 \bbl@ifunset\bbl@##1dflt@}%
4058 {\nameuse{##1family}}%
4059 \bbl@csarg\gdef{WFF@\f@family}}}% Flag
4060 \bbl@exp{\bbl@add\bbl@tempa{* \<##1family>= \f@family\\}%
4061 \space\space\fontname\font\\}%
4062 \bbl@csarg\xdef{##1dflt@}{\f@family}%
4063 \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4064 }%
4065 \ifx\bbl@tempa\@empty\else
4066 \bbl@infowarn{The following font families will use the default\\%
4067 settings for all or some languages:\\%
4068 \bbl@tempa
4069 There is nothing intrinsically wrong with it, but\\%
4070 'babel' will no set Script and Language, which could\\%
4071 be relevant in some languages. If your document uses\\%
4072 these families, consider redefining them with \string\babelfont.\\%
4073 Reported}%
4074 \fi
4075 \endgroup}
4076 \fi
4077 \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.

```

4078 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4079 \bbl@xin@{<>}{#1}%
4080 \ifin@
4081 \bbl@exp{\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4082 \fi
4083 \bbl@exp{%
4084 \def\\#2#1}% eg, \rmdefault{\bbl@rmdflt@lang}
4085 \\bbl@ifsamestring{#2}{\f@family}{\\#3\let\bbl@tempa\relax}}}%
4086 % TODO - next should be global?, but even local does its job. I'm
4087 % still not sure -- must investigate:
4088 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4089 \let\bbl@tempe\bbl@mapselect
4090 \let\bbl@mapselect\relax
4091 \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4092 \let#4\@empty % Make sure \renewfontfamily is valid
4093 \bbl@exp{%
4094 \let\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4095 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4096 {\newfontscript\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4097 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4098 {\newfontlanguage\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4099 \\renewfontfamily\\#4%
4100 [\bbl@cs{lsys@\languagename},#2]}{#3}% ie \bbl@exp{..}{#3}
4101 \begingroup
4102 #4%
4103 \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4104 \endgroup
4105 \let#4\bbl@temp@fam
4106 \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
4107 \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.

```
4108 \def\bbl@font@rst#1#2#3#4{%
4109   \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.

```
4110 \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 :-).

```
4111 \newcommand\babelFSstore[2][]{%
4112   \bbl@ifblank{#1}%
4113   {\bbl@csarg\def{sname@#2}{Latin}}%
4114   {\bbl@csarg\def{sname@#2}{#1}}%
4115   \bbl@provide@dirs{#2}%
4116   \bbl@csarg\ifnum{wdir@#2}>\z@
4117     \let\bbl@beforeforeign\leavevmode
4118     \EnableBabelHook{babel-bidi}%
4119   \fi
4120   \bbl@foreach{#2}{%
4121     \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4122     \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4123     \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4124 \def\bbl@FSstore#1#2#3#4{%
4125   \bbl@csarg\edef{#2default#1}{#3}%
4126   \expandafter\addto\csname extras#1\endcsname{%
4127     \let#4#3%
4128     \ifx#3\f@family
4129       \edef#3{\csname bbl@#2default#1\endcsname}%
4130       \fontfamily{#3}\selectfont
4131     \else
4132       \edef#3{\csname bbl@#2default#1\endcsname}%
4133     \fi}%
4134   \expandafter\addto\csname noextras#1\endcsname{%
4135     \ifx#3\f@family
4136       \fontfamily{#4}\selectfont
4137     \fi
4138     \let#3#4}}
4139 \let\bbl@langfeatures\@empty
4140 \def\babelFSfeatures{% make sure \fontspec is redefined once
4141   \let\bbl@ori@fontspec\fontspec
4142   \renewcommand\fontspec[1][]{%
4143     \bbl@ori@fontspec[\bbl@langfeatures##1]}
4144   \let\babelFSfeatures\bbl@FSfeatures
4145   \babelFSfeatures}
4146 \def\bbl@FSfeatures#1#2{%
4147   \expandafter\addto\csname extras#1\endcsname{%
4148     \babel@save\bbl@langfeatures
4149     \edef\bbl@langfeatures{#2,}}
4150 <</Font selection>>
```

## 13 Hooks for XeTeX and LuaTeX

### 13.1 XeTeX

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

```
4151 <<{*Footnote changes}>> ≡
4152 \bbl@trace{Bidi footnotes}
4153 \ifnum\bbl@bidimode>\z@
4154   \def\bbl@footnote#1#2#3{%
4155     \ifnextchar[%
4156       {\bbl@footnote@o{#1}{#2}{#3}}%
4157       {\bbl@footnote@x{#1}{#2}{#3}}}
4158   \def\bbl@footnote@x#1#2#3#4{%
4159     \bgroup
4160     \select@language@x{\bbl@main@language}%
4161     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4162     \egroup}
4163   \def\bbl@footnote@o#1#2#3[#4]#5{%
4164     \bgroup
4165     \select@language@x{\bbl@main@language}%
4166     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4167     \egroup}
4168   \def\bbl@footnotetext#1#2#3{%
4169     \ifnextchar[%
4170       {\bbl@footnotetext@o{#1}{#2}{#3}}%
4171       {\bbl@footnotetext@x{#1}{#2}{#3}}}
4172   \def\bbl@footnotetext@x#1#2#3#4{%
4173     \bgroup
4174     \select@language@x{\bbl@main@language}%
4175     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4176     \egroup}
4177   \def\bbl@footnotetext@o#1#2#3[#4]#5{%
4178     \bgroup
4179     \select@language@x{\bbl@main@language}%
4180     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4181     \egroup}
4182   \def\BabelFootnote#1#2#3#4{%
4183     \ifx\bbl@fn@footnote\@undefined
4184       \let\bbl@fn@footnote\footnote
4185     \fi
4186     \ifx\bbl@fn@footnotetext\@undefined
4187       \let\bbl@fn@footnotetext\footnotetext
4188     \fi
4189     \bbl@ifblank{#2}%
4190     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4191      \@namedef{\bbl@stripslash#1text}%
4192      {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4193     {\def#1{\bbl@exp{\bbl@footnote{\bbl@foreignlanguage{#2}}}{#3}{#4}}%
4194      \@namedef{\bbl@stripslash#1text}%
4195      {\bbl@exp{\bbl@footnotetext{\bbl@foreignlanguage{#2}}}{#3}{#4}}}%
4196   \fi
4197 <</Footnote changes>>
```

Now, the code.

```
4198 <{*xetex}>
4199 \def\BabelStringsDefault{unicode}
4200 \let\xebbl@stop\relax
```

```

4201 \AddBabelHook{xetex}{encodedcommands}{%
4202   \def\bbl@tempa{#1}%
4203   \ifx\bbl@tempa\@empty
4204     \XeTeXinputencoding"bytes"%
4205   \else
4206     \XeTeXinputencoding"#1"%
4207   \fi
4208   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4209 \AddBabelHook{xetex}{stopcommands}{%
4210   \xebbl@stop
4211   \let\xebbl@stop\relax}
4212 \def\bbl@intraspace#1 #2 #3@@{%
4213   \bbl@csarg\gdef{\xeisp@language}%
4214     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4215 \def\bbl@intrapenalty#1@@{%
4216   \bbl@csarg\gdef{\xeipn@language}%
4217     {\XeTeXlinebreakpenalty #1\relax}}
4218 \def\bbl@provide@intraspace{%
4219   \bbl@xin@{\bbl@cl{\lnbrk}}{s}%
4220   \ifin@else\bbl@xin@{\bbl@cl{\lnbrk}}{c}\fi
4221   \ifin@
4222     \bbl@ifunset{\bbl@intsp@language}{}%
4223     {\expandafter\ifx\cname\bbl@intsp@language\endcsname\@empty\else
4224       \ifx\bbl@KVP@intraspace\@nil
4225         \bbl@exp{%
4226           \\bbl@intraspace\bbl@cl{intsp}\\@@}%
4227         \fi
4228         \ifx\bbl@KVP@intrapenalty\@nil
4229           \bbl@intrapenalty0@@
4230         \fi
4231       \fi
4232       \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4233         \expandafter\bbl@intraspace\bbl@KVP@intraspace@@
4234       \fi
4235       \ifx\bbl@KVP@intrapenalty\@nil\else
4236         \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty@@
4237       \fi
4238       \bbl@exp{%
4239         \\bbl@add\<extras\language>{%
4240           \XeTeXlinebreaklocale "\bbl@cl{lbcpr}"%
4241           \<bbl@xeisp@language>%
4242           \<bbl@xeipn@language>%
4243           \\bbl@toglobal\<extras\language>%
4244           \\bbl@add\<noextras\language>{%
4245             \XeTeXlinebreaklocale "en"%
4246             \\bbl@toglobal\<noextras\language>}}%
4247       \ifx\bbl@ispace\undefined
4248         \gdef\bbl@ispace{\bbl@cl{\xeisp}}%
4249       \ifx\AtBeginDocument\@notprerr
4250         \expandafter\@secondoftwo % to execute right now
4251       \fi
4252       \AtBeginDocument{%
4253         \expandafter\bbl@add
4254         \cselectfont \endcsname{\bbl@ispace}%
4255         \expandafter\bbl@toglobal\cselectfont \endcsname}%
4256     \fi}%
4257 \fi}
4258 \ifx\DisableBabelHook\undefined\endinput\fi
4259 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}

```



```

4260 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfonts}
4261 \DisableBabelHook{babel-fontspec}
4262 <<Font selection>>
4263 \input txtbabel.def
4264 </xetex>

```

## 13.2 Layout

*In progress.*

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

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

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

```

4265 (*texxet)
4266 \providecommand\bbl@provide@intraspace{}
4267 \bbl@trace{Redefinitions for bidi layout}
4268 \def\bbl@sspre@caption{%
4269   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4270 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4271 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4272 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4273 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4274   \def\hangfrom#1{%
4275     \setbox\@tempboxa\hbox{#1}}%
4276     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4277     \noindent\box\@tempboxa}
4278 \def\raggedright{%
4279   \let\@centercr
4280   \bbl@startskip\z@skip
4281   \@rightskip\@flushglue
4282   \bbl@endskip\@rightskip
4283   \parindent\z@
4284   \parfillskip\bbl@startskip}
4285 \def\raggedleft{%
4286   \let\@centercr
4287   \bbl@startskip\@flushglue
4288   \bbl@endskip\z@skip
4289   \parindent\z@
4290   \parfillskip\bbl@endskip}
4291 \fi
4292 \IfBabelLayout{lists}
4293   {\bbl@sreplace\list
4294     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4295     \def\bbl@listleftmargin{%
4296       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4297     \ifcase\bbl@engine
4298       \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4299       \def\p@enumiii{\p@enumii}\theenumii{}
4300     \fi
4301     \bbl@sreplace\@verbatim
4302     {\leftskip\@totalleftmargin}%
4303     {\bbl@startskip\textwidth
4304       \advance\bbl@startskip-\linewidth}%
4305     \bbl@sreplace\@verbatim

```

```

4306     {\rightskip\z@skip}%
4307     {\bbl@endskip\z@skip}}%
4308 {}
4309 \IfBabelLayout{contents}
4310 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4311  \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4312 {}
4313 \IfBabelLayout{columns}
4314 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4315  \def\bbl@outputbox#1{%
4316    \hb@xt@\textwidth{%
4317      \hskip\columnwidth
4318      \hfil
4319      {\normalcolor\vrule \@width\columnseprule}%
4320      \hfil
4321      \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4322      \hskip-\textwidth
4323      \hb@xt@\columnwidth{\box\@outputbox \hss}%
4324      \hskip\columnsep
4325      \hskip\columnwidth}}}%
4326 {}
4327 <<Footnote changes>>
4328 \IfBabelLayout{footnotes}%
4329 {\BabelFootnote\footnote\language{}{}}%
4330  \BabelFootnote\localfootnote\language{}{}}%
4331  \BabelFootnote\mainfootnote{}{}}{}
4332 {}

```

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.

```

4333 \IfBabelLayout{counters}%
4334 {\let\bbl@latinarabic=\@arabic
4335  \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4336  \let\bbl@asciroman=\@roman
4337  \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4338  \let\bbl@asciiRoman=\@Roman
4339  \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
4340 </texxet>

```

### 13.3 LuaTeX

The loader for luatex is based solely on language.dat, which is read on the fly. The code shouldn't be executed when the format is build, so we check if \AddBabelHook is defined. Then comes a modified version of the loader in hyphen.cfg (without the hyphenmins stuff, which is under the direct control of babel).

The names \l@<language> are defined and take some value from the beginning because all ldf files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first selected (which usually means when the ldf finishes). If a language has been loaded, \bbl@hyphendata@<num> exists (with the names of the files read).

The default setup preloads the first language into the format. This is intended mainly for 'english', so that it's available without further intervention from the user. To avoid duplicating it, the following rule applies: if the "0th" language and the first language in language.dat have the same name then just ignore the latter. If there are new synonymous, the are added, but note if the language patterns have not been preloaded they won't at run time.

Other preloaded languages could be read twice, if they have been preloaded into the format. This is not optimal, but it shouldn't happen very often – with luatex patterns are best loaded when the document is typeset, and the “0th” language is preloaded just for backwards compatibility.

As of 1.1b, lua(e)tex is taken into account. Formerly, loading of patterns on the fly didn't work in this format, but with the new loader it does. Unfortunately, the format is not based on babel, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format language.dat is used (under the principle of a single source), instead of language.def.

Of course, there is room for improvements, like tools to read and reassign languages, which would require modifying the language list, and better error handling.

We need catcode tables, but no format (targeted by babel) provide a command to allocate them (although there are packages like ctablestack). FIX - This isn't true anymore. For the moment, a dangerous approach is used - just allocate a high random number and cross the fingers. To complicate things, etex.sty changes the way languages are allocated.

This files is read at three places: (1) when plain.def, babel.sty starts, to read the list of available languages from language.dat (for the base option); (2) at hyphen.cfg, to modify some macros; (3) in the middle of plain.def and babel.sty, by babel.def, with the commands and other definitions for luatex (eg, \babelpatterns).

```

4341 (*luatex)
4342 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4343 \bbl@trace{Read language.dat}
4344 \ifx\bbl@readstream\undefined
4345   \csname newread\endcsname\bbl@readstream
4346 \fi
4347 \begingroup
4348   \toks@{}
4349   \count@ \z@ % 0=start, 1=0th, 2=normal
4350   \def\bbl@process@line#1#2 #3 #4 {%
4351     \ifx=#1%
4352       \bbl@process@synonym{#2}%
4353     \else
4354       \bbl@process@language{#1#2}{#3}{#4}%
4355     \fi
4356     \ignorespaces}
4357   \def\bbl@manylang{%
4358     \ifnum\bbl@last>\@ne
4359       \bbl@info{Non-standard hyphenation setup}%
4360     \fi
4361     \let\bbl@manylang\relax}
4362   \def\bbl@process@language#1#2#3{%
4363     \ifcase\count@
4364       \@ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
4365     \or
4366       \count@\tw@
4367     \fi
4368     \ifnum\count@=\tw@
4369       \expandafter\addlanguage\csname l@#1\endcsname
4370       \language\allocationnumber
4371       \chardef\bbl@last\allocationnumber
4372       \bbl@manylang
4373       \let\bbl@elt\relax
4374       \xdef\bbl@languages{%
4375         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4376     \fi
4377     \the\toks@
4378     \toks@{}}

```

```

4379 \def\bbl@process@synonym@aux#1#2{%
4380   \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4381   \let\bbl@elt\relax
4382   \xdef\bbl@languages{%
4383     \bbl@languages\bbl@elt{#1}{#2}{}}}%
4384 \def\bbl@process@synonym#1{%
4385   \ifcase\count@
4386     \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4387   \or
4388     \@ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4389   \else
4390     \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4391   \fi}
4392 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4393   \chardef\l@english\z@
4394   \chardef\l@USenglish\z@
4395   \chardef\bbl@last\z@
4396   \global\@namedef{\bbl@hyphendata@0}{\hyphen.tex{}}
4397   \gdef\bbl@languages{%
4398     \bbl@elt{english}{0}{\hyphen.tex{}}%
4399     \bbl@elt{USenglish}{0}{}}%
4400 \else
4401   \global\let\bbl@languages@format\bbl@languages
4402   \def\bbl@elt#1#2#3#4{% Remove all except language 0
4403     \ifnum#2>\z@\else
4404       \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4405     \fi}%
4406   \xdef\bbl@languages{\bbl@languages}%
4407 \fi
4408 \def\bbl@elt#1#2#3#4{\@namedef{zth#1}{}} % Define flags
4409 \bbl@languages
4410 \openin\bbl@readstream=language.dat
4411 \ifeof\bbl@readstream
4412   \bbl@warning{I couldn't find language.dat. No additional\\%
4413     patterns loaded. Reported}%
4414 \else
4415   \loop
4416     \endlinechar\m@ne
4417     \read\bbl@readstream to \bbl@line
4418     \endlinechar\^^M
4419     \if T\ifeof\bbl@readstream F\fi T\relax
4420     \ifx\bbl@line\@empty\else
4421       \edef\bbl@line{\bbl@line\space\space\space}%
4422       \expandafter\bbl@process@line\bbl@line\relax
4423     \fi
4424   \repeat
4425 \fi
4426 \endgroup
4427 \bbl@trace{Macros for reading patterns files}
4428 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
4429 \ifx\babelcatcodetablenum\@undefined
4430   \ifx\newcatcodetable\@undefined
4431     \def\babelcatcodetablenum{5211}
4432     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4433   \else
4434     \newcatcodetable\babelcatcodetablenum
4435     \newcatcodetable\bbl@pattcodes
4436   \fi
4437 \else

```

```

4438 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4439 \fi
4440 \def\bbl@luapatterns#1#2{%
4441   \bbl@get@enc#1::\@@@
4442   \setbox\z@\hbox\bgroup
4443     \begingroup
4444       \savecatcodetable\babelcatcodetablenum\relax
4445       \initcatcodetable\bbl@pattcodes\relax
4446       \catcodetable\bbl@pattcodes\relax
4447       \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4448       \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~ =13
4449       \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4450       \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4451       \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4452       \catcode`\'=12 \catcode`\'=12 \catcode`\`=12
4453       \input #1\relax
4454       \catcodetable\babelcatcodetablenum\relax
4455     \endgroup
4456   \def\bbl@tempa{#2}%
4457   \ifx\bbl@tempa@empty\else
4458     \input #2\relax
4459   \fi
4460 \egroup}%
4461 \def\bbl@patterns@lua#1{%
4462   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4463     \csname l@#1\endcsname
4464     \edef\bbl@tempa{#1}%
4465   \else
4466     \csname l@#1:\f@encoding\endcsname
4467     \edef\bbl@tempa{#1:\f@encoding}%
4468   \fi\relax
4469   \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4470   \@ifundefined{bbl@hyphendata@the\language}%
4471   {\def\bbl@elt##1##2##3##4{%
4472     \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4473     \def\bbl@tempb{##3}%
4474     \ifx\bbl@tempb@empty\else % if not a synonymous
4475       \def\bbl@tempc{##3}{##4}%
4476     \fi
4477     \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4478     \fi}%
4479   \bbl@languages
4480   \@ifundefined{bbl@hyphendata@the\language}%
4481   {\bbl@info{No hyphenation patterns were set for\%
4482     language '\bbl@tempa'. Reported}}%
4483   {\expandafter\expandafter\expandafter\bbl@luapatterns
4484     \csname bbl@hyphendata@the\language\endcsname}}}%
4485 \endinput\fi
4486 % Here ends \ifx\AddBabelHook\undefined
4487 % A few lines are only read by hyphen.cfg
4488 \ifx\DisableBabelHook\undefined
4489   \AddBabelHook{luatex}{everylanguage}{%
4490     \def\process@language##1##2##3{%
4491       \def\process@line####1####2 ####3 ####4 {}}%
4492   \AddBabelHook{luatex}{loadpatterns}{%
4493     \input #1\relax
4494     \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4495       {#1}{}}%
4496   \AddBabelHook{luatex}{loadexceptions}{%

```

```

4497 \input #1\relax
4498 \def\bbl@tempb##1##2{{##1}{##2}}%
4499 \expandafter\def\csname bbl@hyphendata@the\language\endcsname
4500 {\expandafter\expandafter\expandafter\bbl@tempb
4501 \csname bbl@hyphendata@the\language\endcsname}}
4502 \endinput\fi
4503 % Here stops reading code for hyphen.cfg
4504 % The following is read the 2nd time it's loaded
4505 \begingroup
4506 \catcode`\%=12
4507 \catcode`\'=12
4508 \catcode`\%=12
4509 \catcode`\:=12
4510 \directlua{
4511   Babel = Babel or {}
4512   function Babel.bytes(line)
4513     return line:gsub(".",
4514       function (chr) return unicode.utf8.char(string.byte(chr)) end)
4515   end
4516   function Babel.begin_process_input()
4517     if luatexbase and luatexbase.add_to_callback then
4518       luatexbase.add_to_callback('process_input_buffer',
4519         Babel.bytes, 'Babel.bytes')
4520     else
4521       Babel.callback = callback.find('process_input_buffer')
4522       callback.register('process_input_buffer', Babel.bytes)
4523     end
4524   end
4525   function Babel.end_process_input ()
4526     if luatexbase and luatexbase.remove_from_callback then
4527       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
4528     else
4529       callback.register('process_input_buffer', Babel.callback)
4530     end
4531   end
4532   function Babel.addpatterns(pp, lg)
4533     local lg = lang.new(lg)
4534     local pats = lang.patterns(lg) or ''
4535     lang.clear_patterns(lg)
4536     for p in pp:gmatch('[^%s]+') do
4537       ss = ''
4538       for i in string.utfcharacters(p:gsub('%d', '')) do
4539         ss = ss .. '%d?' .. i
4540       end
4541       ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
4542       ss = ss:gsub('%.%%d%?$', '%%.')
4543       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4544       if n == 0 then
4545         tex.sprint(
4546           [[\string\csname\space bbl@info\endcsname{New pattern: }
4547           .. p .. [{}]])
4548         pats = pats .. ' ' .. p
4549       else
4550         tex.sprint(
4551           [[\string\csname\space bbl@info\endcsname{Renew pattern: }
4552           .. p .. [{}]])
4553       end
4554     end
4555     lang.patterns(lg, pats)

```

```

4556 end
4557 }
4558 \endgroup
4559 \ifx\newattribute\undefined\else
4560 \newattribute\bbbl@attr@locale
4561 \directlua{ Babel.attr_locale = luatexbase.registernumber'bbbl@attr@locale'}
4562 \AddBabelHook{luatex}{beforeextras}{%
4563   \setattribute\bbbl@attr@locale\localeid}
4564 \fi
4565 \def\BabelStringsDefault{unicode}
4566 \let\luabbbl@stop\relax
4567 \AddBabelHook{luatex}{encodedcommands}{%
4568   \def\bbbl@tempa{utf8}\def\bbbl@tempb{#1}%
4569   \ifx\bbbl@tempa\bbbl@tempb\else
4570     \directlua{Babel.begin_process_input()}%
4571     \def\luabbbl@stop{%
4572       \directlua{Babel.end_process_input()}}%
4573   \fi}%
4574 \AddBabelHook{luatex}{stopcommands}{%
4575   \luabbbl@stop
4576   \let\luabbbl@stop\relax}
4577 \AddBabelHook{luatex}{patterns}{%
4578   \@ifundefined{bbbl@hyphendata@the\language}%
4579   {\def\bbbl@elt##1##2##3##4{%
4580     \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
4581     \def\bbbl@tempb{##3}%
4582     \ifx\bbbl@tempb\empty\else % if not a synonymous
4583       \def\bbbl@tempc{##3}{##4}%
4584     \fi
4585     \bbbl@csarg\xdef{hyphendata@##2}{\bbbl@tempc}%
4586     \fi}%
4587   \bbbl@languages
4588   \@ifundefined{bbbl@hyphendata@the\language}%
4589   {\bbbl@info{No hyphenation patterns were set for\%
4590     language '#2'. Reported}}%
4591   {\expandafter\expandafter\expandafter\bbbl@luapatterns
4592     \csname bbbl@hyphendata@the\language\endcsname}}}%
4593   \@ifundefined{bbbl@patterns@}{}%
4594   \begingroup
4595     \bbbl@xin@{,\number\language,}{,\bbbl@pttnlist}%
4596     \ifin\else
4597       \ifx\bbbl@patterns@\empty\else
4598         \directlua{ Babel.addpatterns(
4599           [[\bbbl@patterns@]], \number\language) }%
4600       \fi
4601       \@ifundefined{bbbl@patterns@#1}%
4602       \empty
4603       {\directlua{ Babel.addpatterns(
4604         [[\space\csname bbbl@patterns@#1\endcsname]],
4605         \number\language) }}%
4606       \xdef\bbbl@pttnlist{\bbbl@pttnlist\number\language,}%
4607     \fi
4608   \endgroup}%
4609   \bbbl@exp{%
4610     \bbbl@ifunset{bbbl@prehc\languagename}{}%
4611     {\bbbl@ifblank{\bbbl@cs{prehc\languagename}}}%
4612     {\prehyphenchar=\bbbl@c1{prehc}\relax}}}%

```

`\babelpatterns` This macro adds patterns. Two macros are used to store them: `\bbbl@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.

```

4613 \@onlypreamble\babelpatterns
4614 \AtEndOfPackage{%
4615   \newcommand\babelpatterns[2][\@empty]{%
4616     \ifx\bbl@patterns@relax
4617       \let\bbl@patterns@\@empty
4618     \fi
4619     \ifx\bbl@pttnlist\@empty\else
4620       \bbl@warning{%
4621         You must not intermingle \string\selectlanguage\space and\%
4622         \string\babelpatterns\space or some patterns will not\%
4623         be taken into account. Reported}%
4624     \fi
4625     \ifx\@empty#1%
4626       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
4627     \else
4628       \edef\bbl@tempb{\zap@space#1 \@empty}%
4629       \bbl@for\bbl@tempa\bbl@tempb{%
4630         \bbl@fixname\bbl@tempa
4631         \bbl@iflanguage\bbl@tempa{%
4632           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
4633             \ifundefined\bbl@patterns@\bbl@tempa}%
4634             \@empty
4635             {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
4636             #2}}}%
4637     \fi}}

```

## 13.4 Southeast Asian scripts

First, some general code for line breaking, used by `\babelposthyphenation`.

*In progress.* Replace regular (ie, implicit) discretionaries by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionaries are not touched.

For the moment, only 3 SA languages are activated by default (see Unicode UAX 14).

```

4638 \directlua{
4639   Babel = Babel or {}
4640   Babel.linebreaking = Babel.linebreaking or {}
4641   Babel.linebreaking.before = {}
4642   Babel.linebreaking.after = {}
4643   Babel.locale = {} % Free to use, indexed with \localeid
4644   function Babel.linebreaking.add_before(func)
4645     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
4646     table.insert(Babel.linebreaking.before , func)
4647   end
4648   function Babel.linebreaking.add_after(func)
4649     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
4650     table.insert(Babel.linebreaking.after , func)
4651   end
4652 }
4653 \def\bbl@intraspace#1 #2 #3\@{
4654   \directlua{
4655     Babel = Babel or {}
4656     Babel.intraspaces = Babel.intraspaces or {}
4657     Babel.intraspaces['\csname bbl@sbcpr@language\endcsname'] = %
4658       {b = #1, p = #2, m = #3}
4659     Babel.locale_props[\the\localeid].intraspace = %

```



```

4660      {b = #1, p = #2, m = #3}
4661  }}
4662 \def\bbl@intrapenalty#1\@@{%
4663   \directlua{
4664     Babel = Babel or {}
4665     Babel.intrapenalties = Babel.intrapenalties or {}
4666     Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
4667     Babel.locale_props[\the\localeid].intrapenalty = #1
4668   }}
4669 \begingroup
4670 \catcode`\%=12
4671 \catcode`\^=14
4672 \catcode`\'=12
4673 \catcode`\~=12
4674 \gdef\bbl@seaintraspace{^
4675   \let\bbl@seaintraspace\relax
4676   \directlua{
4677     Babel = Babel or {}
4678     Babel.sea_enabled = true
4679     Babel.sea_ranges = Babel.sea_ranges or {}
4680     function Babel.set_chranges (script, chrng)
4681       local c = 0
4682       for s, e in string.gmatch(chrng..' ', '(-)%%.(-)%s') do
4683         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
4684         c = c + 1
4685       end
4686     end
4687     function Babel.sea_disc_to_space (head)
4688       local sea_ranges = Babel.sea_ranges
4689       local last_char = nil
4690       local quad = 655360      ^^ 10 pt = 655360 = 10 * 65536
4691       for item in node.traverse(head) do
4692         local i = item.id
4693         if i == node.id'glyph' then
4694           last_char = item
4695         elseif i == 7 and item.subtype == 3 and last_char
4696           and last_char.char > 0x0C99 then
4697           quad = font.getfont(last_char.font).size
4698           for lg, rg in pairs(sea_ranges) do
4699             if last_char.char > rg[1] and last_char.char < rg[2] then
4700               lg = lg:sub(1, 4) ^^ Remove trailing number of, eg, Cyl1
4701               local intraspace = Babel.intraspaces[lg]
4702               local intrapenalty = Babel.intrapenalties[lg]
4703               local n
4704               if intrapenalty ~= 0 then
4705                 n = node.new(14, 0) ^^ penalty
4706                 n.penalty = intrapenalty
4707                 node.insert_before(head, item, n)
4708               end
4709               n = node.new(12, 13) ^^ (glue, spaceskip)
4710               node.setglue(n, intraspace.b * quad,
4711                 intraspace.p * quad,
4712                 intraspace.m * quad)
4713               node.insert_before(head, item, n)
4714               node.remove(head, item)
4715             end
4716           end
4717         end
4718       end

```

```

4719     end
4720 }^^
4721 \bbl@luahyphenate}
4722 \catcode`\%=14
4723 \gdef\bbl@cjkintraspacespace{%
4724   \let\bbl@cjkintraspacespace\relax
4725   \directlua{
4726     Babel = Babel or {}
4727     require'babel-data-cjk.lua'
4728     Babel.cjk_enabled = true
4729     function Babel.cjk_linebreak(head)
4730       local GLYPH = node.id'glyph'
4731       local last_char = nil
4732       local quad = 655360      % 10 pt = 655360 = 10 * 65536
4733       local last_class = nil
4734       local last_lang = nil
4735
4736       for item in node.traverse(head) do
4737         if item.id == GLYPH then
4738
4739           local lang = item.lang
4740
4741           local LOCALE = node.get_attribute(item,
4742             luatexbase.registernumber'bbl@attr@locale')
4743           local props = Babel.locale_props[LOCALE]
4744
4745           local class = Babel.cjk_class[item.char].c
4746
4747           if class == 'cp' then class = 'cl' end % )] as CL
4748           if class == 'id' then class = 'I' end
4749
4750           local br = 0
4751           if class and last_class and Babel.cjk_breaks[last_class][class] then
4752             br = Babel.cjk_breaks[last_class][class]
4753           end
4754
4755           if br == 1 and props.linebreak == 'c' and
4756             lang ~= \the\l@nohyphenation\space and
4757             last_lang ~= \the\l@nohyphenation then
4758             local intrapenalty = props.intrapenalty
4759             if intrapenalty ~= 0 then
4760               local n = node.new(14, 0)      % penalty
4761               n.penalty = intrapenalty
4762               node.insert_before(head, item, n)
4763             end
4764             local intraspacespace = props.intraspacespace
4765             local n = node.new(12, 13)      % (glue, spaceskip)
4766             node.setglue(n, intraspacespace.b * quad,
4767               intraspacespace.p * quad,
4768               intraspacespace.m * quad)
4769             node.insert_before(head, item, n)
4770           end
4771
4772           quad = font.getfont(item.font).size
4773           last_class = class
4774           last_lang = lang
4775         else % if penalty, glue or anything else
4776           last_class = nil
4777         end

```

```

4778     end
4779     lang.hyphenate(head)
4780 end
4781 }%
4782 \bbl@luahyphenate}
4783 \gdef\bbl@luahyphenate{%
4784 \let\bbl@luahyphenate\relax
4785 \directlua{
4786   luatexbase.add_to_callback('hyphenate',
4787   function (head, tail)
4788     if Babel.linebreaking.before then
4789       for k, func in ipairs(Babel.linebreaking.before) do
4790         func(head)
4791       end
4792     end
4793     if Babel.cjk_enabled then
4794       Babel.cjk_linebreak(head)
4795     end
4796     lang.hyphenate(head)
4797     if Babel.linebreaking.after then
4798       for k, func in ipairs(Babel.linebreaking.after) do
4799         func(head)
4800       end
4801     end
4802     if Babel.sea_enabled then
4803       Babel.sea_disc_to_space(head)
4804     end
4805   end,
4806   'Babel.hyphenate')
4807 }
4808 }
4809 \endgroup
4810 \def\bbl@provide@intraspace{%
4811 \bbl@ifunset{\bbl@intsp@{language}}{%
4812   {\expandafter\ifx\csname\bbl@intsp@{language}\endcsname\@empty\else
4813     \bbl@xin@{\bbl@cl{lbrk}}{c}%
4814     \ifin@           % cjk
4815     \bbl@cjk_intraspace
4816     \directlua{
4817       Babel = Babel or {}
4818       Babel.locale_props = Babel.locale_props or {}
4819       Babel.locale_props[\the\localeid].linebreak = 'c'
4820     }%
4821     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}{\@}%
4822     \ifx\bbl@KVP@intrapenalty\@nil
4823       \bbl@intrapenalty0\@@
4824     \fi
4825   \else           % sea
4826     \bbl@sea_intraspace
4827     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}{\@}%
4828     \directlua{
4829       Babel = Babel or {}
4830       Babel.sea_ranges = Babel.sea_ranges or {}
4831       Babel.set_chranges('\bbl@cl{sbc} ',
4832         '\bbl@cl{chrng}')
4833     }%
4834     \ifx\bbl@KVP@intrapenalty\@nil
4835       \bbl@intrapenalty0\@@
4836     \fi

```

```

4837     \fi
4838     \fi
4839     \ifx\bbbl@KVP@intrapenalty\@nil\else
4840         \expandafter\bbbl@intrapenalty\bbbl@KVP@intrapenalty\@@
4841     \fi}}

```

### 13.5 CJK line breaking

Minimal line breaking for CJK scripts, mainly intended for simple documents and short texts as a secondary language. Only line breaking, with a little stretching for justification, without any attempt to adjust the spacing. It is based on (but does not strictly follow) the Unicode algorithm.

We first need a little table with the corresponding line breaking properties. A few characters have an additional key for the width (fullwidth vs. halfwidth), not yet used. There is a separate file, defined below.

*Work in progress.*

Common stuff.

```

4842 \AddBabelHook{babel-fontspec}{afterextras}{\bbbl@switchfont}
4843 \AddBabelHook{babel-fontspec}{beforestart}{\bbbl@ckeckstdfonts}
4844 \DisableBabelHook{babel-fontspec}
4845 <<Font selection>>

```

### 13.6 Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a short function which just traverse the node list to carry out the replacements. The table `loc_to_scr` gets the locale 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.

```

4846 \directlua{
4847 Babel.script_blocks = {
4848   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
4849             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
4850   ['Armn'] = {{0x0530, 0x058F}},
4851   ['Beng'] = {{0x0980, 0x09FF}},
4852   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
4853   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
4854   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
4855             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
4856   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
4857   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
4858             {0xAB00, 0xAB2F}},
4859   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
4860   % Don't follow strictly Unicode, which places some Coptic letters in
4861   % the 'Greek and Coptic' block
4862   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
4863   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
4864             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
4865             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
4866             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
4867             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
4868             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
4869   ['Hebr'] = {{0x0590, 0x05FF}},

```

```

4870 ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
4871           {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
4872 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
4873 ['Knda'] = {{0x0C80, 0x0CFF}},
4874 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
4875           {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
4876           {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
4877 ['Lao'] = {{0x0E80, 0x0EFF}},
4878 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
4879           {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
4880           {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
4881 ['Mahj'] = {{0x11150, 0x1117F}},
4882 ['Mlym'] = {{0x0D00, 0x0D7F}},
4883 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
4884 ['Orya'] = {{0x0B00, 0x0B7F}},
4885 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
4886 ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
4887 ['Taml'] = {{0x0B80, 0x0BFF}},
4888 ['Telu'] = {{0x0C00, 0x0C7F}},
4889 ['Tfng'] = {{0x2D30, 0x2D7F}},
4890 ['Thai'] = {{0x0E00, 0x0E7F}},
4891 ['Tibt'] = {{0x0F00, 0x0FFF}},
4892 ['Vaii'] = {{0xA500, 0xA63F}},
4893 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
4894 }
4895
4896 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
4897 Babel.script_blocks.Hant = Babel.script_blocks.Hans
4898 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
4899
4900 function Babel.locale_map(head)
4901   if not Babel.locale_mapped then return head end
4902
4903   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
4904   local GLYPH = node.id('glyph')
4905   local inmath = false
4906   local toloc_save
4907   for item in node.traverse(head) do
4908     local toloc
4909     if not inmath and item.id == GLYPH then
4910       % Optimization: build a table with the chars found
4911       if Babel.chr_to_loc[item.char] then
4912         toloc = Babel.chr_to_loc[item.char]
4913       else
4914         for lc, maps in pairs(Babel.loc_to_scr) do
4915           for _, rg in pairs(maps) do
4916             if item.char >= rg[1] and item.char <= rg[2] then
4917               Babel.chr_to_loc[item.char] = lc
4918               toloc = lc
4919               break
4920             end
4921           end
4922         end
4923       end
4924       % Now, take action, but treat composite chars in a different
4925       % fashion, because they 'inherit' the previous locale. Not yet
4926       % optimized.
4927       if not toloc and
4928         (item.char >= 0x0300 and item.char <= 0x036F) or

```

```

4929         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
4930         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
4931         toloc = toloc_save
4932     end
4933     if toloc and toloc > -1 then
4934         if Babel.locale_props[toloc].lg then
4935             item.lang = Babel.locale_props[toloc].lg
4936             node.set_attribute(item, LOCALE, toloc)
4937         end
4938         if Babel.locale_props[toloc]['/'..item.font] then
4939             item.font = Babel.locale_props[toloc]['/'..item.font]
4940         end
4941         toloc_save = toloc
4942     end
4943     elseif not inmath and item.id == 7 then
4944         item.replace = item.replace and Babel.locale_map(item.replace)
4945         item.pre      = item.pre and Babel.locale_map(item.pre)
4946         item.post     = item.post and Babel.locale_map(item.post)
4947     elseif item.id == node.id'math' then
4948         inmath = (item.subtype == 0)
4949     end
4950 end
4951 return head
4952 end
4953 }

```

The code for `\babelcharproperty` is straightforward. Just note the modified lua table can be different.

```

4954 \newcommand\babelcharproperty[1]{%
4955   \count@=#1\relax
4956   \ifvmode
4957     \expandafter\bbl@chprop
4958   \else
4959     \bbl@error{\string\babelcharproperty\space can be used only in\%
4960       vertical mode (preamble or between paragraphs)}%
4961     {See the manual for futher info}%
4962   \fi}
4963 \newcommand\bbl@chprop[3][\the\count@]{%
4964   \@tempcnta=#1\relax
4965   \bbl@ifunset{\bbl@chprop@#2}%
4966   {\bbl@error{No property named '#2'. Allowed values are\%
4967     direction (bc), mirror (bmg), and linebreak (lb)}%
4968     {See the manual for futher info}}%
4969   }%
4970   \loop
4971     \bbl@cs{chprop@#2}{#3}%
4972   \ifnum\count@<\@tempcnta
4973     \advance\count@\@ne
4974   \repeat}
4975 \def\bbl@chprop@direction#1{%
4976   \directlua{
4977     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
4978     Babel.characters[\the\count@]['d'] = '#1'
4979   }}
4980 \let\bbl@chprop@bc\bbl@chprop@direction
4981 \def\bbl@chprop@mirror#1{%
4982   \directlua{
4983     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
4984     Babel.characters[\the\count@]['m'] = '\number#1'

```

```

4985 }}
4986 \let\bbl@chprop@bmg\bbl@chprop@mirror
4987 \def\bbl@chprop@linebreak#1{%
4988   \directlua{
4989     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
4990     Babel.cjk_characters[\the\count@]['c'] = '#1'
4991   }}
4992 \let\bbl@chprop@lb\bbl@chprop@linebreak
4993 \def\bbl@chprop@locale#1{%
4994   \directlua{
4995     Babel.chr_to_loc = Babel.chr_to_loc or {}
4996     Babel.chr_to_loc[\the\count@] =
4997       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
4998   }}

```

Post-handling hyphenation patterns for non-standard rules, like ff to ff-f. There are still some issues with speed (not very slow, but still slow).

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: `str_to_nodes` converts the string returned by a function to a node list, taking the node at base as a model (font, language, etc.); `fetch_word` fetches a series of glyphs and discretionaries, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck). `post_hyphenate_replace` is the callback applied after `lang.hyphenate`. This means the automatic hyphenation points are known. As empty captures return a byte position (as explained in the `luatex` manual), we must convert it to a utf8 position. With `first`, the last byte can be the leading byte in a utf8 sequence, so we just remove it and add 1 to the resulting length. With `last` we must take into account the capture position points to the next character. Here `word_head` points to the starting node of the text to be matched.

```

4999 \begingroup
5000 \catcode`\#=12
5001 \catcode`\%=12
5002 \catcode`\&=14
5003 \directlua{
5004   Babel.linebreaking.post_replacements = {}
5005   Babel.linebreaking.pre_replacements = {}
5006
5007   function Babel.str_to_nodes(fn, matches, base)
5008     local n, head, last
5009     if fn == nil then return nil end
5010     for s in string.utfvalues(fn(matches)) do
5011       if base.id == 7 then
5012         base = base.replace
5013       end
5014       n = node.copy(base)
5015       n.char = s
5016       if not head then
5017         head = n
5018       else
5019         last.next = n
5020       end
5021       last = n
5022     end
5023     return head
5024   end
5025
5026   function Babel.fetch_word(head, funct)
5027     local word_string = ''
5028     local word_nodes = {}

```

```

5029     local lang
5030     local item = head
5031     local inmath = false
5032
5033     while item do
5034
5035         if item.id == 29
5036             and not(item.char == 124) && ie, not |
5037             and not(item.char == 61) && ie, not =
5038             and not inmath
5039             and (item.lang == lang or lang == nil) then
5040             lang = lang or item.lang
5041             word_string = word_string .. unicode.utf8.char(item.char)
5042             word_nodes[#word_nodes+1] = item
5043
5044         elseif item.id == 7 and item.subtype == 2 and not inmath then
5045             word_string = word_string .. '='
5046             word_nodes[#word_nodes+1] = item
5047
5048         elseif item.id == 7 and item.subtype == 3 and not inmath then
5049             word_string = word_string .. '|'
5050             word_nodes[#word_nodes+1] = item
5051
5052         elseif item.id == 11 and item.subtype == 0 then
5053             inmath = true
5054
5055         elseif word_string == '' then
5056             && pass
5057
5058         else
5059             return word_string, word_nodes, item, lang
5060         end
5061
5062         item = item.next
5063     end
5064 end
5065
5066 function Babel.post_hyphenate_replace(head)
5067     local u = unicode.utf8
5068     local lbkr = Babel.linebreaking.post_replacements
5069     local word_head = head
5070
5071     while true do
5072         local w, wn, nw, lang = Babel.fetch_word(word_head)
5073         if not lang then return head end
5074
5075         if not lbkr[lang] then
5076             break
5077         end
5078
5079         for k=1, #lbkr[lang] do
5080             local p = lbkr[lang][k].pattern
5081             local r = lbkr[lang][k].replace
5082
5083             while true do
5084                 local matches = { u.match(w, p) }
5085                 if #matches < 2 then break end
5086
5087                 local first = table.remove(matches, 1)

```



```

5088     local last = table.remove(matches, #matches)
5089
5090     %% Fix offsets, from bytes to unicode.
5091     first = u.len(w:sub(1, first-1)) + 1
5092     last = u.len(w:sub(1, last-1))
5093
5094     local new %% used when inserting and removing nodes
5095     local changed = 0
5096
5097     %% This loop traverses the replace list and takes the
5098     %% corresponding actions
5099     for q = first, last do
5100         local crep = r[q-first+1]
5101         local char_node = wn[q]
5102         local char_base = char_node
5103
5104         if crep and crep.data then
5105             char_base = wn[crep.data+first-1]
5106         end
5107
5108         if crep == {} then
5109             break
5110         elseif crep == nil then
5111             changed = changed + 1
5112             node.remove(head, char_node)
5113         elseif crep and (crep.pre or crep.no or crep.post) then
5114             changed = changed + 1
5115             d = node.new(7, 0) %% (disc, discretionary)
5116             d.pre = Babel.str_to_nodes(crep.pre, matches, char_base)
5117             d.post = Babel.str_to_nodes(crep.post, matches, char_base)
5118             d.replace = Babel.str_to_nodes(crep.no, matches, char_base)
5119             d.attr = char_base.attr
5120             if crep.pre == nil then %% TeXbook p96
5121                 d.penalty = crep.penalty or tex.hyphenpenalty
5122             else
5123                 d.penalty = crep.penalty or tex.exhyphenpenalty
5124             end
5125             head, new = node.insert_before(head, char_node, d)
5126             node.remove(head, char_node)
5127             if q == 1 then
5128                 word_head = new
5129             end
5130         elseif crep and crep.string then
5131             changed = changed + 1
5132             local str = crep.string(matches)
5133             if str == '' then
5134                 if q == 1 then
5135                     word_head = char_node.next
5136                 end
5137                 head, new = node.remove(head, char_node)
5138             elseif char_node.id == 29 and u.len(str) == 1 then
5139                 char_node.char = string.utfvalue(str)
5140             else
5141                 local n
5142                 for s in string.utfvalues(str) do
5143                     if char_node.id == 7 then
5144                         log('Automatic hyphens cannot be replaced, just removed.')
5145                     else
5146                         n = node.copy(char_base)

```

```

5147         end
5148         n.char = s
5149         if q == 1 then
5150             head, new = node.insert_before(head, char_node, n)
5151             word_head = new
5152         else
5153             node.insert_before(head, char_node, n)
5154         end
5155     end
5156
5157     node.remove(head, char_node)
5158     end %% string length
5159     end %% if char and char.string
5160     end %% for char in match
5161     if changed > 20 then
5162         texio.write('Too many changes. Ignoring the rest.')
5163     elseif changed > 0 then
5164         w, wn, nw = Babel.fetch_word(word_head)
5165     end
5166
5167     end %% for match
5168     end %% for patterns
5169     word_head = nw
5170 end %% for words
5171 return head
5172 end
5173
5174 %%%
5175 %% Preliminary code for \babelprehyphenation
5176 %% TODO. Copypaste pattern. Merge with fetch_word
5177 function Babel.fetch_subtext(head, funct)
5178     local word_string = ''
5179     local word_nodes = {}
5180     local lang
5181     local item = head
5182     local inmath = false
5183
5184     while item do
5185
5186         if item.id == 29 then
5187             local locale = node.get_attribute(item, Babel.attr_locale)
5188
5189             if not(item.char == 124) %% ie, not | = space
5190                 and not inmath
5191                 and (locale == lang or lang == nil) then
5192                 lang = lang or locale
5193                 word_string = word_string .. unicode.utf8.char(item.char)
5194                 word_nodes[#word_nodes+1] = item
5195             end
5196
5197             if item == node.tail(head) then
5198                 item = nil
5199                 return word_string, word_nodes, item, lang
5200             end
5201
5202             elseif item.id == 12 and item.subtype == 13 and not inmath then
5203                 word_string = word_string .. '|'
5204                 word_nodes[#word_nodes+1] = item
5205

```

```

5206         if item == node.tail(head) then
5207             item = nil
5208             return word_string, word_nodes, item, lang
5209         end
5210
5211         elseif item.id == 11 and item.subtype == 0 then
5212             inmath = true
5213
5214         elseif word_string == '' then
5215             %% pass
5216
5217         else
5218             return word_string, word_nodes, item, lang
5219         end
5220
5221         item = item.next
5222     end
5223 end
5224
5225 &% TODO. Copypaste pattern. Merge with pre_hyphenate_replace
5226 function Babel.pre_hyphenate_replace(head)
5227     local u = unicode.utf8
5228     local lbkr = Babel.linebreaking.pre_replacements
5229     local word_head = head
5230
5231     while true do
5232         local w, wn, nw, lang = Babel.fetch_subtext(word_head)
5233         if not lang then return head end
5234
5235         if not lbkr[lang] then
5236             break
5237         end
5238
5239         for k=1, #lbkr[lang] do
5240             local p = lbkr[lang][k].pattern
5241             local r = lbkr[lang][k].replace
5242
5243             while true do
5244                 local matches = { u.match(w, p) }
5245                 if #matches < 2 then break end
5246
5247                 local first = table.remove(matches, 1)
5248                 local last = table.remove(matches, #matches)
5249
5250                 &% Fix offsets, from bytes to unicode.
5251                 first = u.len(w:sub(1, first-1)) + 1
5252                 last = u.len(w:sub(1, last-1))
5253
5254                 local new &% used when inserting and removing nodes
5255                 local changed = 0
5256
5257                 &% This loop traverses the replace list and takes the
5258                 &% corresponding actions
5259                 for q = first, last do
5260                     local crep = r[q-first+1]
5261                     local char_node = wn[q]
5262                     local char_base = char_node
5263
5264                     if crep and crep.data then

```

```

5265         char_base = wn[crep.data+first-1]
5266     end
5267
5268     if crep == {} then
5269         break
5270     elseif crep == nil then
5271         changed = changed + 1
5272         node.remove(head, char_node)
5273     elseif crep and crep.string then
5274         changed = changed + 1
5275         local str = crep.string(matches)
5276         if str == '' then
5277             if q == 1 then
5278                 word_head = char_node.next
5279             end
5280             head, new = node.remove(head, char_node)
5281         elseif char_node.id == 29 and u.len(str) == 1 then
5282             char_node.char = string.utfvalue(str)
5283         else
5284             local n
5285             for s in string.utfvalues(str) do
5286                 if char_node.id == 7 then
5287                     log('Automatic hyphens cannot be replaced, just removed.')
5288                 else
5289                     n = node.copy(char_base)
5290                 end
5291                 n.char = s
5292                 if q == 1 then
5293                     head, new = node.insert_before(head, char_node, n)
5294                     word_head = new
5295                 else
5296                     node.insert_before(head, char_node, n)
5297                 end
5298             end
5299
5300             node.remove(head, char_node)
5301         end &% string length
5302     end &% if char and char.string
5303 end &% for char in match
5304 if changed > 20 then
5305     texio.write('Too many changes. Ignoring the rest.')
5306 elseif changed > 0 then
5307     &% For one-to-one can we modify directly the
5308     &% values without re-fetching? Very likely.
5309     w, wn, nw = Babel.fetch_subtext(word_head)
5310 end
5311
5312     end &% for match
5313 end &% for patterns
5314 word_head = nw
5315 end &% for words
5316 return head
5317 end
5318 &%% end of preliminary code for \babelprehyphenation
5319
5320 &% The following functions belong to the next macro
5321
5322 &% This table stores capture maps, numbered consecutively
5323 Babel.capture_maps = {}

```

```

5324
5325 function Babel.capture_func(key, cap)
5326   local ret = "[" .. cap:gsub('{{([0-9])}}', "]]..m[%1]..[" .. "]"
5327   ret = ret:gsub('{{([0-9])|([^\]|+)|(.-)}}', Babel.capture_func_map)
5328   ret = ret:gsub("%[%[%]%%.%.%", '')
5329   ret = ret:gsub("%.%.%[%[%]%%", '')
5330   return key .. "[=function(m) return ]] .. ret .. [[ end]]
5331 end
5332
5333 function Babel.capt_map(from, mapno)
5334   return Babel.capture_maps[mapno][from] or from
5335 end
5336
5337 &% Handle the {n|abc|ABC} syntax in captures
5338 function Babel.capture_func_map(capno, from, to)
5339   local froms = {}
5340   for s in string.utfcharacters(from) do
5341     table.insert(froms, s)
5342   end
5343   local cnt = 1
5344   table.insert(Babel.capture_maps, {})
5345   local mlen = table.getn(Babel.capture_maps)
5346   for s in string.utfcharacters(to) do
5347     Babel.capture_maps[mlen][froms[cnt]] = s
5348     cnt = cnt + 1
5349   end
5350   return "]]..Babel.capt_map(m[" .. capno .. "], " ..
5351     (mlen) .. ").." .. "["
5352 end
5353 }

```

Now the  $\text{\TeX}$  high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the  $\{n\}$  syntax. For example,  $\text{pre}=\{1\}\{1\}$ - becomes `function(m) return m[1]..m[1]..'-' end`, where  $m$  are the matches returned after applying the pattern. With a mapped capture the functions are similar to `function(m) return Babel.capt_map(m[1], 1) end`, where the last argument identifies the mapping to be applied to  $m[1]$ . The way it is carried out is somewhat tricky, but the effect is not dissimilar to `lua load` – save the code as string in a  $\text{\TeX}$  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).

```

5354 \catcode`\#=6
5355 \gdef\babelposthyphenation#1#2#3{&%
5356   \bbl@activateposthyphen
5357   \begingroup
5358     \def\babeltempa{\bbl@add@list\babeltempb}&%
5359     \let\babeltempb\@empty
5360     \bbl@foreach{#3}{&%
5361       \bbl@ifsamestring{##1}{remove}&%
5362       {\bbl@add@list\babeltempb{nil}}&%
5363       {\directlua{
5364         local rep = [[##1]]
5365         rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
5366         rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5367         rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
5368         rep = rep:gsub(' (string)%s*=%s*([^\s,]*)', Babel.capture_func)
5369         tex.print([[ \string\babeltempa{[]] .. rep .. [[]]])
5370       }}}&%

```

```

5371 \directlua{
5372     local lbkr = Babel.linebreaking.post_replacements
5373     local u = unicode.utf8
5374     %% Convert pattern:
5375     local patt = string.gsub(#[#2]=, '%s', '')
5376     if not u.find(patt, '()', nil, true) then
5377         patt = '()' .. patt .. '()'
5378     end
5379     patt = u.gsub(patt, '{(.)}',
5380         function (n)
5381             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5382         end)
5383     lbkr[\the\csname l@#1\endcsname] = lbkr[\the\csname l@#1\endcsname] or {}
5384     table.insert(lbkr[\the\csname l@#1\endcsname],
5385         { pattern = patt, replace = { \babeltempb } })
5386 }%%
5387 \endgroup}
5388 % TODO. Working !!! Copypaste pattern.
5389 \gdef\babelprehyphenation#1#2#3{%%
5390     \bbl@activateprehyphen
5391     \begingroup
5392         \def\babeltempa{\bbl@add@list\babeltempb}%%
5393         \let\babeltempb\@empty
5394         \bbl@foreach{#3}{%%
5395             \bbl@ifsamestring{##1}{remove}%%
5396             {\bbl@add@list\babeltempb{nil}}%%
5397             {\directlua{
5398                 local rep = [#{##1}]
5399                 rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5400                 tex.print([[\string\babeltempa{#}] .. rep .. [{}]])
5401             }}%%
5402         \directlua{
5403             local lbkr = Babel.linebreaking.pre_replacements
5404             local u = unicode.utf8
5405             %% Convert pattern:
5406             local patt = string.gsub(#[#2]=, '%s', '')
5407             if not u.find(patt, '()', nil, true) then
5408                 patt = '()' .. patt .. '()'
5409             end
5410             patt = u.gsub(patt, '{(.)}',
5411                 function (n)
5412                     return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5413                 end)
5414             lbkr[\the\csname bbl@id@@#1\endcsname] = lbkr[\the\csname bbl@id@@#1\endcsname] or {}
5415             table.insert(lbkr[\the\csname bbl@id@@#1\endcsname],
5416                 { pattern = patt, replace = { \babeltempb } })
5417         }%%
5418     \endgroup}
5419 \endgroup
5420 \def\bbl@activateposthyphen{%
5421     \let\bbl@activateposthyphen\relax
5422     \directlua{
5423         Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5424     }}
5425 % TODO. Working !!!
5426 \def\bbl@activateprehyphen{%
5427     \let\bbl@activateprehyphen\relax
5428     \directlua{
5429         Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)

```

```
5430  }}
```

## 13.7 Layout

Unlike xetex, luatex requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with `bidi=basic`, without having to patch almost any macro where text direction is relevant.

`\@hangfrom` is useful in many contexts and it is redefined always with the `layout` option.

There are, however, a number of issues when the text direction is not the same as the box direction (as set by `\bodydir`), and when `\parbox` and `\hangindent` are involved.

Fortunately, latest releases of luatex simplify a lot the solution with `\shapemode`.

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, `tabular` seems to work (at least in simple cases) with `array`, `tabularx`, `hline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```
5431 \bbl@trace{Redefinitions for bidi layout}
5432 \ifx\@eqnnum\@undefined\else
5433   \ifx\bbl@attr@dir\@undefined\else
5434     \edef\@eqnnum{%
5435       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
5436       \unexpanded\expandafter{\@eqnnum}}}%
5437   \fi
5438 \fi
5439 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
5440 \ifnum\bbl@bidimode>\z@
5441   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
5442     \bbl@exp{%
5443       \mathdir\the\bodydir
5444       #1%           Once entered in math, set boxes to restore values
5445       \<ifmmode>%
5446       \everyvbox{%
5447         \the\everyvbox
5448         \bodydir\the\bodydir
5449         \mathdir\the\mathdir
5450         \everyhbox{\the\everyhbox}%
5451         \everyvbox{\the\everyvbox}}%
5452       \everyhbox{%
5453         \the\everyhbox
5454         \bodydir\the\bodydir
5455         \mathdir\the\mathdir
5456         \everyhbox{\the\everyhbox}%
5457         \everyvbox{\the\everyvbox}}%
5458       \<fi>}}%
5459   \def\@hangfrom#1{%
5460     \setbox\@tempboxa\hbox{{#1}}%
5461     \hangindent\wd\@tempboxa
5462     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5463       \shapemode\@ne
5464     \fi
5465     \noindent\box\@tempboxa}
5466 \fi
5467 \IfBabelLayout{tabular}
5468   {\let\bbl@OL@tabular\@tabular
5469     \bbl@replace\@tabular{$}\bbl@nextfake$}%
5470   \let\bbl@NL@tabular\@tabular
5471   \AtBeginDocument{%
```

```

5472 \ifx\bb1@NL@@tabular\@tabular\else
5473 \bb1@replace\@tabular{$}\bb1@nextfake$}%
5474 \let\bb1@NL@@tabular\@tabular
5475 \fi}}
5476 {}
5477 \IfBabelLayout{lists}
5478 {\let\bb1@OL@list\list
5479 \bb1@sreplace\list{\parshape}\bb1@listparshape}%
5480 \let\bb1@NL@list\list
5481 \def\bb1@listparshape#1#2#3{%
5482 \parshape #1 #2 #3 %
5483 \ifnum\bb1@getluadir{page}=\bb1@getluadir{par}\else
5484 \shapemode\tw@
5485 \fi}}
5486 {}
5487 \IfBabelLayout{graphics}
5488 {\let\bb1@pictresetdir\relax
5489 \def\bb1@pictsetdir{%
5490 \ifcase\bb1@thetextdir
5491 \let\bb1@pictresetdir\relax
5492 \else
5493 \textdir TLT\relax
5494 \def\bb1@pictresetdir{\textdir TRT\relax}%
5495 \fi}%
5496 \let\bb1@OL@@picture\@picture
5497 \let\bb1@OL@put\put
5498 \bb1@sreplace\@picture{\hskip-}\bb1@pictsetdir\hskip-}%
5499 \def\put(#1,#2)#3{% Not easy to patch. Better redefine.
5500 \@killglue
5501 \raise#2\unitlength
5502 \hb@xt@z{\kern#1\unitlength\bb1@pictresetdir#3}\hss}}%
5503 \AtBeginDocument
5504 {\ifx\tikz@atbegin@node\undefined\else
5505 \let\bb1@OL@pgfpicture\pgfpicture
5506 \bb1@sreplace\pgfpicture{\pgfpicturetrue}\bb1@pictsetdir\pgfpicturetrue}%
5507 \bb1@add\pgfsys@beginpicture{\bb1@pictsetdir}%
5508 \bb1@add\tikz@atbegin@node{\bb1@pictresetdir}%
5509 \fi}}
5510 {}

```

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.

```

5511 \IfBabelLayout{counters}%
5512 {\let\bb1@OL@@textsuperscript\@textsuperscript
5513 \bb1@sreplace\@textsuperscript{\m@th}\m@th\mathdir\pagedir}%
5514 \let\bb1@latinarabic=\@arabic
5515 \let\bb1@OL@@arabic\@arabic
5516 \def\@arabic#1{\babelsublr{\bb1@latinarabic#1}}%
5517 \@ifpackagewith{babel}{bidi=default}%
5518 {\let\bb1@asciroman=\@roman
5519 \let\bb1@OL@@roman\@roman
5520 \def\@roman#1{\babelsublr{\ensureascii{\bb1@asciroman#1}}}%
5521 \let\bb1@asciiRoman=\@Roman
5522 \let\bb1@OL@@roman\@Roman
5523 \def\@Roman#1{\babelsublr{\ensureascii{\bb1@asciiRoman#1}}}%
5524 \let\bb1@OL@labelenumii\labelenumii
5525 \def\labelenumii{}\theenumii}%
5526 \let\bb1@OL@p@enumiii\p@enumiii

```



```

5527 \def\p@enumiii{\p@enumii}\theenumii{}{}{}{}
5528 <<Footnote changes>>
5529 \IfBabelLayout{footnotes}%
5530 {\let\bbl@OL@footnote\footnote
5531 \BabelFootnote\footnote\languagename{}{}%
5532 \BabelFootnote\localfootnote\languagename{}{}%
5533 \BabelFootnote\mainfootnote{}{}{}{}
5534 {}

```

Some  $\text{\LaTeX}$  macros use internally the math mode for text formatting. They have very little in common and are grouped here, as a single option.

```

5535 \IfBabelLayout{extras}%
5536 {\let\bbl@OL@underline\underline
5537 \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
5538 \let\bbl@OL@LaTeX2e\LaTeX2e
5539 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
5540 \if b\expandafter\@car\@series\@nil\boldmath\fi
5541 \babelsublr}%
5542 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}
5543 {}
5544 </luatex>

```

### 13.8 Auto bidi with basic and basic-r

The file `babel-data-bidi.lua` currently only contains data. It is a large and boring file and it is not shown here (see the generated file), but here is a sample:

```

[0x25]={d='et'},
[0x26]={d='on'},
[0x27]={d='on'},
[0x28]={d='on', m=0x29},
[0x29]={d='on', m=0x28},
[0x2A]={d='on'},
[0x2B]={d='es'},
[0x2C]={d='cs'},

```

For the meaning of these codes, see the Unicode standard.

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

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

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, *what* they do and *why*, and not only *how*), but I think (or I hope) I've managed to understand them.

In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually *two* R modes (set explicitly in Unicode with RLM and ALM). In `babel` the `dir` is set by a higher protocol based on the language/script, which in turn sets the correct `dir` (<l>, <r> or <al>).

From UAX#9: “Where available, markup should be used instead of the explicit formatting characters”. So, this simple version just ignores formatting characters. Actually, most of that annex is devoted to how to handle them.

BD14-BD16 are not implemented. Unicode (and the W3C) are making a great effort to deal with some special problematic cases in “streamed” plain text. I don’t think this is the way to go – particular issues should be fixed by a high level interface taking into account the needs of the document. And here is where luatex excels, because everything related to bidi writing is under our control.

```

5545 (*basic-r)
5546 Babel = Babel or {}
5547
5548 Babel.bidi_enabled = true
5549
5550 require('babel-data-bidi.lua')
5551
5552 local characters = Babel.characters
5553 local ranges = Babel.ranges
5554
5555 local DIR = node.id("dir")
5556
5557 local function dir_mark(head, from, to, outer)
5558   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
5559   local d = node.new(DIR)
5560   d.dir = '+' .. dir
5561   node.insert_before(head, from, d)
5562   d = node.new(DIR)
5563   d.dir = '-' .. dir
5564   node.insert_after(head, to, d)
5565 end
5566
5567 function Babel.bidi(head, ispar)
5568   local first_n, last_n      -- first and last char with nums
5569   local last_es              -- an auxiliary 'last' used with nums
5570   local first_d, last_d      -- first and last char in L/R block
5571   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):

```

5572   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
5573   local strong_lr = (strong == 'l') and 'l' or 'r'
5574   local outer = strong
5575
5576   local new_dir = false
5577   local first_dir = false
5578   local inmath = false
5579
5580   local last_lr
5581
5582   local type_n = ''
5583
5584   for item in node.traverse(head) do
5585
5586     -- three cases: glyph, dir, otherwise
5587     if item.id == node.id'glyph'
5588       or (item.id == 7 and item.subtype == 2) then
5589
5590       local itemchar

```

```

5591     if item.id == 7 and item.subtype == 2 then
5592         itemchar = item.replace.char
5593     else
5594         itemchar = item.char
5595     end
5596     local chardata = characters[itemchar]
5597     dir = chardata and chardata.d or nil
5598     if not dir then
5599         for nn, et in ipairs(ranges) do
5600             if itemchar < et[1] then
5601                 break
5602             elseif itemchar <= et[2] then
5603                 dir = et[3]
5604                 break
5605             end
5606         end
5607     end
5608     dir = dir or 'l'
5609     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).

```

5610     if new_dir then
5611         attr_dir = 0
5612         for at in node.traverse(item.attr) do
5613             if at.number == luatexbase.registernumber'bbl@attr@dir' then
5614                 attr_dir = at.value % 3
5615             end
5616         end
5617         if attr_dir == 1 then
5618             strong = 'r'
5619         elseif attr_dir == 2 then
5620             strong = 'al'
5621         else
5622             strong = 'l'
5623         end
5624         strong_lr = (strong == 'l') and 'l' or 'r'
5625         outer = strong_lr
5626         new_dir = false
5627     end
5628
5629     if dir == 'nsm' then dir = strong end -- W1

```

**Numbers.** The dual <al>/<r> system for R is somewhat cumbersome.

```

5630     dir_real = dir -- We need dir_real to set strong below
5631     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:

```

5632     if strong == 'al' then
5633         if dir == 'en' then dir = 'an' end -- W2
5634         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
5635         strong_lr = 'r' -- W3
5636     end

```

Once finished the basic setup for glyphs, consider the two other cases: dir node and the rest.

```

5637 elseif item.id == node.id'dir' and not inmath then
5638     new_dir = true
5639     dir = nil
5640 elseif item.id == node.id'math' then
5641     inmath = (item.subtype == 0)
5642 else
5643     dir = nil          -- Not a char
5644 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>.

```

5645 if dir == 'en' or dir == 'an' or dir == 'et' then
5646     if dir ~= 'et' then
5647         type_n = dir
5648     end
5649     first_n = first_n or item
5650     last_n = last_es or item
5651     last_es = nil
5652 elseif dir == 'es' and last_n then -- W3+W6
5653     last_es = item
5654 elseif dir == 'cs' then          -- it's right - do nothing
5655     if first_n then -- & if dir = any but en, et, an, es, cs, inc nil
5656         if strong_lr == 'r' and type_n ~= '' then
5657             dir_mark(head, first_n, last_n, 'r')
5658         elseif strong_lr == 'l' and first_d and type_n == 'an' then
5659             dir_mark(head, first_n, last_n, 'r')
5660             dir_mark(head, first_d, last_d, outer)
5661             first_d, last_d = nil, nil
5662         elseif strong_lr == 'l' and type_n ~= '' then
5663             last_d = last_n
5664         end
5665         type_n = ''
5666         first_n, last_n = nil, nil
5667     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:

```

5668 if dir == 'l' or dir == 'r' then
5669     if dir ~= outer then
5670         first_d = first_d or item
5671         last_d = item
5672     elseif first_d and dir ~= strong_lr then
5673         dir_mark(head, first_d, last_d, outer)
5674         first_d, last_d = nil, nil
5675     end
5676 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.

```

5677   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
5678       item.char = characters[item.char] and
5679           characters[item.char].m or item.char
5680   elseif (dir or new_dir) and last_lr ~= item then
5681       local mir = outer .. strong_lr .. (dir or outer)
5682       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
5683           for ch in node.traverse(node.next(last_lr)) do
5684               if ch == item then break end
5685               if ch.id == node.id'glyph' and characters[ch.char] then
5686                   ch.char = characters[ch.char].m or ch.char
5687               end
5688           end
5689       end
5690   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).

```

5691   if dir == 'l' or dir == 'r' then
5692       last_lr = item
5693       strong = dir_real          -- Don't search back - best save now
5694       strong_lr = (strong == 'l') and 'l' or 'r'
5695   elseif new_dir then
5696       last_lr = nil
5697   end
5698 end

```

Mirror the last chars if they are no directed. And make sure any open block is closed, too.

```

5699   if last_lr and outer == 'r' then
5700       for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
5701           if characters[ch.char] then
5702               ch.char = characters[ch.char].m or ch.char
5703           end
5704       end
5705   end
5706   if first_n then
5707       dir_mark(head, first_n, last_n, outer)
5708   end
5709   if first_d then
5710       dir_mark(head, first_d, last_d, outer)
5711   end

```

In boxes, the dir node could be added before the original head, so the actual head is the previous node.

```

5712   return node.prev(head) or head
5713 end
5714 </basic-r>

```

And here the Lua code for bidi=basic:

```

5715 <(*basic)
5716 Babel = Babel or {}
5717
5718 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
5719
5720 Babel.fontmap = Babel.fontmap or {}
5721 Babel.fontmap[0] = {}      -- 1

```

```

5722 Babel.fontmap[1] = {}      -- r
5723 Babel.fontmap[2] = {}      -- al/an
5724
5725 Babel.bidi_enabled = true
5726 Babel.mirroring_enabled = true
5727
5728 require('babel-data-bidi.lua')
5729
5730 local characters = Babel.characters
5731 local ranges = Babel.ranges
5732
5733 local DIR = node.id('dir')
5734 local GLYPH = node.id('glyph')
5735
5736 local function insert_implicit(head, state, outer)
5737   local new_state = state
5738   if state.sim and state.eim and state.sim ~= state.eim then
5739     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
5740     local d = node.new(DIR)
5741     d.dir = '+' .. dir
5742     node.insert_before(head, state.sim, d)
5743     local d = node.new(DIR)
5744     d.dir = '-' .. dir
5745     node.insert_after(head, state.eim, d)
5746   end
5747   new_state.sim, new_state.eim = nil, nil
5748   return head, new_state
5749 end
5750
5751 local function insert_numeric(head, state)
5752   local new
5753   local new_state = state
5754   if state.san and state.ean and state.san ~= state.ean then
5755     local d = node.new(DIR)
5756     d.dir = '+TLT'
5757     _, new = node.insert_before(head, state.san, d)
5758     if state.san == state.sim then state.sim = new end
5759     local d = node.new(DIR)
5760     d.dir = '-TLT'
5761     _, new = node.insert_after(head, state.ean, d)
5762     if state.ean == state.eim then state.eim = new end
5763   end
5764   new_state.san, new_state.ean = nil, nil
5765   return head, new_state
5766 end
5767
5768 -- TODO - \hbox with an explicit dir can lead to wrong results
5769 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
5770 -- was s made to improve the situation, but the problem is the 3-dir
5771 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
5772 -- well.
5773
5774 function Babel.bidi(head, ispar, hdir)
5775   local d -- d is used mainly for computations in a loop
5776   local prev_d = ''
5777   local new_d = false
5778
5779   local nodes = {}
5780   local outer_first = nil

```

```

5781 local inmath = false
5782
5783 local glue_d = nil
5784 local glue_i = nil
5785
5786 local has_en = false
5787 local first_et = nil
5788
5789 local ATDIR = luatexbase.registernumber'bbl@attr@dir'
5790
5791 local save_outer
5792 local temp = node.get_attribute(head, ATDIR)
5793 if temp then
5794     temp = temp % 3
5795     save_outer = (temp == 0 and 'l') or
5796                 (temp == 1 and 'r') or
5797                 (temp == 2 and 'al')
5798 elseif ispar then -- Or error? Shouldn't happen
5799     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
5800 else -- Or error? Shouldn't happen
5801     save_outer = ('TRT' == hdir) and 'r' or 'l'
5802 end
5803 -- when the callback is called, we are just _after_ the box,
5804 -- and the textdir is that of the surrounding text
5805 -- if not ispar and hdir ~= tex.textdir then
5806 --     save_outer = ('TRT' == hdir) and 'r' or 'l'
5807 -- end
5808 local outer = save_outer
5809 local last = outer
5810 -- 'al' is only taken into account in the first, current loop
5811 if save_outer == 'al' then save_outer = 'r' end
5812
5813 local fontmap = Babel.fontmap
5814
5815 for item in node.traverse(head) do
5816
5817     -- In what follows, #node is the last (previous) node, because the
5818     -- current one is not added until we start processing the neutrals.
5819
5820     -- three cases: glyph, dir, otherwise
5821     if item.id == GLYPH
5822         or (item.id == 7 and item.subtype == 2) then
5823
5824         local d_font = nil
5825         local item_r
5826         if item.id == 7 and item.subtype == 2 then
5827             item_r = item.replace -- automatic discs have just 1 glyph
5828         else
5829             item_r = item
5830         end
5831         local chardata = characters[item_r.char]
5832         d = chardata and chardata.d or nil
5833         if not d or d == 'nsm' then
5834             for nn, et in ipairs(ranges) do
5835                 if item_r.char < et[1] then
5836                     break
5837                 elseif item_r.char <= et[2] then
5838                     if not d then d = et[3]
5839                     elseif d == 'nsm' then d_font = et[3]

```

```

5840         end
5841         break
5842     end
5843 end
5844 end
5845 d = d or 'l'
5846
5847 -- A short 'pause' in bidi for mapfont
5848 d_font = d_font or d
5849 d_font = (d_font == 'l' and 0) or
5850           (d_font == 'nsm' and 0) or
5851           (d_font == 'r' and 1) or
5852           (d_font == 'al' and 2) or
5853           (d_font == 'an' and 2) or nil
5854 if d_font and fontmap and fontmap[d_font][item_r.font] then
5855     item_r.font = fontmap[d_font][item_r.font]
5856 end
5857
5858 if new_d then
5859     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
5860     if inmath then
5861         attr_d = 0
5862     else
5863         attr_d = node.get_attribute(item, ATDIR)
5864         attr_d = attr_d % 3
5865     end
5866     if attr_d == 1 then
5867         outer_first = 'r'
5868         last = 'r'
5869     elseif attr_d == 2 then
5870         outer_first = 'r'
5871         last = 'al'
5872     else
5873         outer_first = 'l'
5874         last = 'l'
5875     end
5876     outer = last
5877     has_en = false
5878     first_et = nil
5879     new_d = false
5880 end
5881
5882 if glue_d then
5883     if (d == 'l' and 'l' or 'r') ~= glue_d then
5884         table.insert(nodes, {glue_i, 'on', nil})
5885     end
5886     glue_d = nil
5887     glue_i = nil
5888 end
5889
5890 elseif item.id == DIR then
5891     d = nil
5892     new_d = true
5893
5894 elseif item.id == node.id'glue' and item.subtype == 13 then
5895     glue_d = d
5896     glue_i = item
5897     d = nil
5898

```



```

5899     elseif item.id == node.id'math' then
5900         inmath = (item.subtype == 0)
5901
5902     else
5903         d = nil
5904     end
5905
5906     -- AL <= EN/ET/ES      -- W2 + W3 + W6
5907     if last == 'al' and d == 'en' then
5908         d = 'an'          -- W3
5909     elseif last == 'al' and (d == 'et' or d == 'es') then
5910         d = 'on'          -- W6
5911     end
5912
5913     -- EN + CS/ES + EN      -- W4
5914     if d == 'en' and #nodes >= 2 then
5915         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
5916             and nodes[#nodes-1][2] == 'en' then
5917             nodes[#nodes][2] = 'en'
5918         end
5919     end
5920
5921     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
5922     if d == 'an' and #nodes >= 2 then
5923         if (nodes[#nodes][2] == 'cs')
5924             and nodes[#nodes-1][2] == 'an' then
5925             nodes[#nodes][2] = 'an'
5926         end
5927     end
5928
5929     -- ET/EN                -- W5 + W7->l / W6->on
5930     if d == 'et' then
5931         first_et = first_et or (#nodes + 1)
5932     elseif d == 'en' then
5933         has_en = true
5934         first_et = first_et or (#nodes + 1)
5935     elseif first_et then    -- d may be nil here !
5936         if has_en then
5937             if last == 'l' then
5938                 temp = 'l'    -- W7
5939             else
5940                 temp = 'en'   -- W5
5941             end
5942         else
5943             temp = 'on'       -- W6
5944         end
5945         for e = first_et, #nodes do
5946             if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
5947         end
5948         first_et = nil
5949         has_en = false
5950     end
5951
5952     if d then
5953         if d == 'al' then
5954             d = 'r'
5955             last = 'al'
5956         elseif d == 'l' or d == 'r' then
5957             last = d

```

```

5958     end
5959     prev_d = d
5960     table.insert(nodes, {item, d, outer_first})
5961 end
5962
5963     outer_first = nil
5964
5965 end
5966
5967 -- TODO -- repeated here in case EN/ET is the last node. Find a
5968 -- better way of doing things:
5969 if first_et then      -- dir may be nil here !
5970     if has_en then
5971         if last == 'l' then
5972             temp = 'l'    -- W7
5973         else
5974             temp = 'en'    -- W5
5975         end
5976     else
5977         temp = 'on'        -- W6
5978     end
5979     for e = first_et, #nodes do
5980         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
5981     end
5982 end
5983
5984 -- dummy node, to close things
5985 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
5986
5987 ----- NEUTRAL -----
5988
5989 outer = save_outer
5990 last = outer
5991
5992 local first_on = nil
5993
5994 for q = 1, #nodes do
5995     local item
5996
5997     local outer_first = nodes[q][3]
5998     outer = outer_first or outer
5999     last = outer_first or last
6000
6001     local d = nodes[q][2]
6002     if d == 'an' or d == 'en' then d = 'r' end
6003     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
6004
6005     if d == 'on' then
6006         first_on = first_on or q
6007     elseif first_on then
6008         if last == d then
6009             temp = d
6010         else
6011             temp = outer
6012         end
6013         for r = first_on, q - 1 do
6014             nodes[r][2] = temp
6015             item = nodes[r][1]    -- MIRRORING
6016             if Babel.mirroring_enabled and item.id == GLYPH

```

```

6017         and temp == 'r' and characters[item.char] then
6018             local font_mode = font.fonts[item.font].properties.mode
6019             if font_mode ~= 'harf' and font_mode ~= 'plug' then
6020                 item.char = characters[item.char].m or item.char
6021             end
6022         end
6023     end
6024     first_on = nil
6025 end
6026
6027     if d == 'r' or d == 'l' then last = d end
6028 end
6029
6030 ----- IMPLICIT, REORDER -----
6031
6032 outer = save_outer
6033 last = outer
6034
6035 local state = {}
6036 state.has_r = false
6037
6038 for q = 1, #nodes do
6039
6040     local item = nodes[q][1]
6041
6042     outer = nodes[q][3] or outer
6043
6044     local d = nodes[q][2]
6045
6046     if d == 'nsm' then d = last end           -- W1
6047     if d == 'en' then d = 'an' end
6048     local isdir = (d == 'r' or d == 'l')
6049
6050     if outer == 'l' and d == 'an' then
6051         state.san = state.san or item
6052         state.ean = item
6053     elseif state.san then
6054         head, state = insert_numeric(head, state)
6055     end
6056
6057     if outer == 'l' then
6058         if d == 'an' or d == 'r' then        -- im -> implicit
6059             if d == 'r' then state.has_r = true end
6060             state.sim = state.sim or item
6061             state.eim = item
6062         elseif d == 'l' and state.sim and state.has_r then
6063             head, state = insert_implicit(head, state, outer)
6064         elseif d == 'l' then
6065             state.sim, state.eim, state.has_r = nil, nil, false
6066         end
6067     else
6068         if d == 'an' or d == 'l' then
6069             if nodes[q][3] then -- nil except after an explicit dir
6070                 state.sim = item -- so we move sim 'inside' the group
6071             else
6072                 state.sim = state.sim or item
6073             end
6074             state.eim = item
6075         elseif d == 'r' and state.sim then

```

```

6076         head, state = insert_implicit(head, state, outer)
6077     elseif d == 'r' then
6078         state.sim, state.eim = nil, nil
6079     end
6080 end
6081
6082 if isdir then
6083     last = d          -- Don't search back - best save now
6084 elseif d == 'on' and state.san then
6085     state.san = state.san or item
6086     state.ean = item
6087 end
6088
6089 end
6090
6091 return node.prev(head) or head
6092 end
6093 </basic>

```

## 14 Data for CJK

It is a boring file and it is not shown here (see the generated file), but here is a sample:

```

[0x0021]={c='ex'},
[0x0024]={c='pr'},
[0x0025]={c='po'},
[0x0028]={c='op'},
[0x0029]={c='cp'},
[0x002B]={c='pr'},

```

For the meaning of these codes, see the Unicode standard.

## 15 The ‘nil’ language

This ‘language’ does nothing, except setting the hyphenation patterns to nohyphenation. For this language currently no special definitions are needed or available.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```

6094 <nil>
6095 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language]
6096 \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.

```

6097 \ifx\l@nil\@undefined
6098   \newlanguage\l@nil
6099   \@namedef{bbl@hyphendata@the\l@nil}{\{}}{\{}}% Remove warning
6100   \let\bbl@elt\relax
6101   \edef\bbl@languages{% Add it to the list of languages
6102     \bbl@languages\bbl@elt{nil}{the\l@nil}{\{}}{\{}}
6103 \fi

```

This macro is used to store the values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```

6104 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

The next step consists of defining commands to switch to (and from) the ‘nil’ language.

```
\captionnil
\datenil 6105 \let\captionnil\@empty
6106 \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.

```
6107 \ldf@finish{nil}
6108 </nil>
```

## 16 Support for Plain $\text{\TeX}$ (plain.def)

### 16.1 Not renaming hyphen.tex

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based  $\text{\TeX}$ -format. When asked he responded:

That file name is “sacred”, and if anybody changes it they will cause severe upward/downward compatibility headaches.

People can have a file `locallyhyphen.tex` or whatever they like, but they mustn’t diddle with `hyphen.tex` (or `plain.tex` except to preload additional fonts).

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to achieve the desired effect, based on the `babel` package. If you load each of them with  $\text{\TeX}$ , you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`. As these files are going to be read as the first thing  $\text{\TeX}$  sees, we need to set some category codes just to be able to change the definition of `\input`.

```
6109 (*bplain | blplain)
6110 \catcode`\{=1 % left brace is begin-group character
6111 \catcode`\}=2 % right brace is end-group character
6112 \catcode`\#=6 % hash mark is macro parameter character
```

If a file called `hyphen.cfg` can be found, we make sure that *it* will be read instead of the file `hyphen.tex`. We do this by first saving the original meaning of `\input` (and I use a one letter control sequence for that so as not to waste multi-letter control sequence on this in the format).

```
6113 \openin 0 hyphen.cfg
6114 \ifeof0
6115 \else
6116   \let\input
```

Then `\input` is defined to forget about its argument and load `hyphen.cfg` instead. Once that’s done the original meaning of `\input` can be restored and the definition of `\a` can be forgotten.

```
6117 \def\input #1 {%
6118   \let\input\input
6119   \a hyphen.cfg
6120   \let\input\undefined
6121 }
6122 \fi
6123 </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`.

```
6124 \bplain\la plain.tex
6125 \bplain\la lplain.tex
```

Finally we change the contents of `\fmtname` to indicate that this is *not* the plain format, but a format based on plain with the `babel` package preloaded.

```
6126 \bplain\def\fmtname{babel-plain}
6127 \bplain\def\fmtname{babel-lplain}
```

When you are using a different format, based on `plain.tex` you can make a copy of `blplain.tex`, rename it and replace `plain.tex` with the name of your format file.

## 16.2 Emulating some $\text{\LaTeX}$ features

The following code duplicates or emulates parts of  $\text{\LaTeX} 2_{\epsilon}$  that are needed for `babel`.

```
6128 \langle *Emulate LaTeX \rangle \equiv
6129 % == Code for plain ==
6130 \def\@empty{}
6131 \def\loadlocalcfg#1{%
6132   \openin0#1.cfg
6133   \ifeof0
6134     \closein0
6135   \else
6136     \closein0
6137     {\immediate\write16{*****}%
6138      \immediate\write16{* Local config file #1.cfg used}%
6139      \immediate\write16{*}%
6140     }
6141     \input #1.cfg\relax
6142   \fi
6143   \@endoflfd}
```

## 16.3 General tools

A number of  $\text{\LaTeX}$  macro's that are needed later on.

```
6144 \long\def\@firstofone#1{#1}
6145 \long\def\@firstoftwo#1#2{#1}
6146 \long\def\@secondoftwo#1#2{#2}
6147 \def\@nnil{\@nil}
6148 \def\@gobbletwo#1#2{}
6149 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
6150 \def\@star@or@long#1{%
6151   \@ifstar
6152   {\let\l@ngrel@x\relax#1}%
6153   {\let\l@ngrel@x\long#1}}
6154 \let\l@ngrel@x\relax
6155 \def\@car#1#2\@nil{#1}
6156 \def\@cdr#1#2\@nil{#2}
6157 \let\@typeset@protect\relax
6158 \let\protected@edef\edef
6159 \long\def\@gobble#1{}
6160 \edef\@backslashchar{\expandafter\@gobble\string\}
6161 \def\strip@prefix#1>{}
6162 \def\g@addto@macro#1#2{{%
6163   \toks@\expandafter{#1#2}%
6164   \xdef#1{\the\toks@}}}
```

```

6165 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
6166 \def\@nameuse#1{\csname #1\endcsname}
6167 \def\@ifundefined#1{%
6168   \expandafter\ifx\csname#1\endcsname\relax
6169     \expandafter\@firstoftwo
6170   \else
6171     \expandafter\@secondoftwo
6172   \fi}
6173 \def\@expandtwoargs#1#2#3{%
6174   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
6175 \def\zap@space#1 #2{%
6176   #1%
6177   \ifx#2\@empty\else\expandafter\zap@space\fi
6178   #2}
6179 \let\bbl@trace\@gobble

```

$\text{\LaTeX 2}_{\epsilon}$  has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

6180 \ifx\@preamblecmds\@undefined
6181   \def\@preamblecmds{}
6182 \fi
6183 \def\@onlypreamble#1{%
6184   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
6185     \@preamblecmds\do#1}}
6186 \@onlypreamble\@onlypreamble

```

Mimick  $\text{\LaTeX}$ 's `\AtBeginDocument`; for this to work the user needs to add `\begin{document}` to his file.

```

6187 \def\begin{document}{%
6188   \@begin{document}hook
6189   \global\let\@begin{document}hook\@undefined
6190   \def\do##1{\global\let##1\@undefined}%
6191   \@preamblecmds
6192   \global\let\do\noexpand}
6193 \ifx\@begin{document}hook\@undefined
6194   \def\@begin{document}hook{}
6195 \fi
6196 \@onlypreamble\@begin{document}hook
6197 \def\AtBeginDocument{\g@addto@macro\@begin{document}hook}

```

We also have to mimick  $\text{\LaTeX}$ 's `\AtEndOfPackage`. Our replacement macro is much simpler; it stores its argument in `\@endofldf`.

```

6198 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
6199 \@onlypreamble\AtEndOfPackage
6200 \def\@endofldf{}
6201 \@onlypreamble\@endofldf
6202 \let\bbl@afterlang\@empty
6203 \chardef\bbl@opt@hyphenmap\z@

```

$\text{\LaTeX}$  needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default. There is a trick to hide some conditional commands from the outer `\ifx`. The same trick is applied below.

```

6204 \catcode`\&=\z@
6205 \ifx&\if@files\@undefined
6206   \expandafter\let\csname if@files\endcsname
6207   \csname iffalse\endcsname
6208 \fi
6209 \catcode`\&=4

```

Mimick L<sup>A</sup>T<sub>E</sub>X's commands to define control sequences.

```

6210 \def\newcommand{\@star@or@long\new@command}
6211 \def\new@command#1{%
6212   \@testopt{\@newcommand#1}0}
6213 \def\@newcommand#1[#2]{%
6214   \@ifnextchar [{\@xargdef#1[#2]]%
6215     {\@argdef#1[#2]}}
6216 \long\def\@argdef#1[#2]#3{%
6217   \@yargdef#1\@ne{#2}{#3}}
6218 \long\def\@xargdef#1[#2][#3]#4{%
6219   \expandafter\def\expandafter#1\expandafter{%
6220     \expandafter\@protected@testopt\expandafter #1%
6221     \csname\string#1\expandafter\endcsname{#3}}%
6222   \expandafter\@yargdef \csname\string#1\endcsname
6223   \tw@{#2}{#4}}
6224 \long\def\@yargdef#1#2#3{%
6225   \@tempcnta#3\relax
6226   \advance \@tempcnta \@ne
6227   \let\@hash@\relax
6228   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
6229   \@tempcntb #2%
6230   \@whilenum\@tempcntb <\@tempcnta
6231   \do{%
6232     \edef\reserved@a{\reserved@a\@hash@the\@tempcntb}%
6233     \advance\@tempcntb \@ne}%
6234   \let\@hash@###
6235   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
6236 \def\providecommand{\@star@or@long\provide@command}
6237 \def\provide@command#1{%
6238   \begingroup
6239     \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
6240   \endgroup
6241   \expandafter\ifundefined\@gtempa
6242     {\def\reserved@a{\new@command#1}}%
6243     {\let\reserved@a\relax
6244       \def\reserved@a{\new@command\reserved@a}}%
6245   \reserved@a}%
6246 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
6247 \def\declare@robustcommand#1{%
6248   \edef\reserved@a{\string#1}%
6249   \def\reserved@b{#1}%
6250   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
6251   \edef#1{%
6252     \ifx\reserved@a\reserved@b
6253       \noexpand\x@protect
6254       \noexpand#1%
6255     \fi
6256     \noexpand\protect
6257     \expandafter\noexpand\csname
6258       \expandafter\@gobble\string#1 \endcsname
6259   }%
6260   \expandafter\new@command\csname
6261     \expandafter\@gobble\string#1 \endcsname
6262 }
6263 \def\x@protect#1{%
6264   \ifx\protect\@typeset@protect\else
6265     \@x@protect#1%
6266   \fi

```



```

6267 }
6268 \catcode`\&=\z@ % Trick to hide conditionals
6269 \def\@x@protect#1&fi#2#3{&fi\protect#1}

```

The following little macro `\in@` is taken from `latex.ltx`; it checks whether its first argument is part of its second argument. It uses the boolean `\in@`; allocating a new boolean inside conditionally executed code is not possible, hence the construct with the temporary definition of `\bbl@tempa`.

```

6270 \def\bbl@tempa{\csname newif\endcsname&fin@}
6271 \catcode`\&=4
6272 \ifx\in@\@undefined
6273 \def\in@#1#2{%
6274   \def\in@@##1#1##2##3\in@{%
6275     \ifx\in@@##2\in@false\else\in@true\fi}%
6276   \in@@#2#1\in@\in@@}
6277 \else
6278 \let\bbl@tempa\@empty
6279 \fi
6280 \bbl@tempa

```

$\text{\LaTeX}$  has a macro to check whether a certain package was loaded with specific options. The command has two extra arguments which are code to be executed in either the true or false case. This is used to detect whether the document needs one of the accents to be activated (activegrave and activeacute). For plain  $\text{\TeX}$  we assume that the user wants them to be active by default. Therefore the only thing we do is execute the third argument (the code for the true case).

```

6281 \def\@ifpackagewith#1#2#3#4{#3}

```

The  $\text{\LaTeX}$  macro `\@ifl@aded` checks whether a file was loaded. This functionality is not needed for plain  $\text{\TeX}$  but we need the macro to be defined as a no-op.

```

6282 \def\@ifl@aded#1#2#3#4{}

```

For the following code we need to make sure that the commands `\newcommand` and `\providecommand` exist with some sensible definition. They are not fully equivalent to their  $\text{\LaTeX} 2_{\epsilon}$  versions; just enough to make things work in plain  $\text{\TeX}$  environments.

```

6283 \ifx\@tempcnta\@undefined
6284 \csname newcount\endcsname\@tempcnta\relax
6285 \fi
6286 \ifx\@tempcntb\@undefined
6287 \csname newcount\endcsname\@tempcntb\relax
6288 \fi

```

To prevent wasting two counters in  $\text{\LaTeX} 2.09$  (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```

6289 \ifx\bye\@undefined
6290 \advance\count10 by -2\relax
6291 \fi
6292 \ifx\@ifnextchar\@undefined
6293 \def\@ifnextchar#1#2#3{%
6294   \let\reserved@d=#1%
6295   \def\reserved@a{#2}\def\reserved@b{#3}%
6296   \futurelet\@let@token\@ifnch}
6297 \def\@ifnch{%
6298   \ifx\@let@token\@sptoken
6299     \let\reserved@c\@xifnch
6300   \else
6301     \ifx\@let@token\reserved@d
6302       \let\reserved@c\reserved@a

```

```

6303     \else
6304     \let\reserved@c\reserved@b
6305     \fi
6306 \fi
6307 \reserved@c}
6308 \def\:\let\sptoken= } \: % this makes \sptoken a space token
6309 \def\:\@xifnch} \expandafter\def\:\{\futurelet\@let@token\@ifnch}
6310 \fi
6311 \def\@testopt#1#2{%
6312   \ifnextchar[{\#1}{\#1[#2]}}
6313 \def\@protected@testopt#1{%
6314   \ifx\protect\@typeset@protect
6315     \expandafter\@testopt
6316   \else
6317     \@x@protect#1%
6318   \fi}
6319 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
6320   #2\relax}\fi}
6321 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
6322   \else\expandafter\@gobble\fi{#1}}

```

## 16.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain  $\TeX$  environment.

```

6323 \def\DeclareTextCommand{%
6324   \@dec@text@cmd\providecommand
6325 }
6326 \def\ProvideTextCommand{%
6327   \@dec@text@cmd\providecommand
6328 }
6329 \def\DeclareTextSymbol#1#2#3{%
6330   \@dec@text@cmd\chardef#1{#2}#3\relax
6331 }
6332 \def\@dec@text@cmd#1#2#3{%
6333   \expandafter\def\expandafter#2%
6334     \expandafter{%
6335       \csname#3-cmd\expandafter\endcsname
6336       \expandafter#2%
6337       \csname#3\string#2\endcsname
6338     }%
6339 %   \let\@ifdefinable\rc@ifdefinable
6340   \expandafter#1\csname#3\string#2\endcsname
6341 }
6342 \def\@current@cmd#1{%
6343   \ifx\protect\@typeset@protect\else
6344     \noexpand#1\expandafter\@gobble
6345   \fi
6346 }
6347 \def\@changed@cmd#1#2{%
6348   \ifx\protect\@typeset@protect
6349     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
6350       \expandafter\ifx\csname ?\string#1\endcsname\relax
6351         \expandafter\def\csname ?\string#1\endcsname{%
6352           \@changed@x@err{#1}%
6353         }%
6354       \fi
6355     \global\expandafter\let
6356     \csname\cf@encoding \string#1\expandafter\endcsname

```

```

6357         \csname ?\string#1\endcsname
6358     \fi
6359     \csname\cf@encoding\string#1%
6360         \expandafter\endcsname
6361 \else
6362     \noexpand#1%
6363 \fi
6364 }
6365 \def\@changed@x@err#1{%
6366     \errhelp{Your command will be ignored, type <return> to proceed}%
6367     \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
6368 \def\DeclareTextCommandDefault#1{%
6369     \DeclareTextCommand#1?%
6370 }
6371 \def\ProvideTextCommandDefault#1{%
6372     \ProvideTextCommand#1?%
6373 }
6374 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
6375 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
6376 \def\DeclareTextAccent#1#2#3{%
6377     \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
6378 }
6379 \def\DeclareTextCompositeCommand#1#2#3#4{%
6380     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
6381     \edef\reserved@b{\string##1}%
6382     \edef\reserved@c{%
6383         \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
6384     \ifx\reserved@b\reserved@c
6385         \expandafter\expandafter\expandafter\ifx
6386             \expandafter\@car\reserved@a\relax\relax\@nil
6387             \@text@composite
6388     \else
6389         \edef\reserved@b##1{%
6390             \def\expandafter\noexpand
6391                 \csname#2\string#1\endcsname####1{%
6392                 \noexpand\@text@composite
6393                 \expandafter\noexpand\csname#2\string#1\endcsname
6394                 ####1\noexpand\@empty\noexpand\@text@composite
6395                 {##1}%
6396             }%
6397         }%
6398         \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
6399     \fi
6400     \expandafter\def\csname\expandafter\string\csname
6401         #2\endcsname\string#1-\string#3\endcsname{#4}
6402 \else
6403     \errhelp{Your command will be ignored, type <return> to proceed}%
6404     \errmessage{\string\DeclareTextCompositeCommand\space used on
6405         inappropriate command \protect#1}
6406 \fi
6407 }
6408 \def\@text@composite#1#2#3\@text@composite{%
6409     \expandafter\@text@composite@x
6410     \csname\string#1-\string#2\endcsname
6411 }
6412 \def\@text@composite@x#1#2{%
6413     \ifx#1\relax
6414         #2%
6415     \else

```

```

6416      #1%
6417      \fi
6418 }
6419 %
6420 \def\@strip@args#1:#2-#3\@strip@args{#2}
6421 \def\DeclareTextComposite#1#2#3#4{%
6422     \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
6423     \bgroup
6424         \lccode` \@=#4%
6425         \lowercase{%
6426     \egroup
6427     \reserved@a @%
6428 }%
6429 }
6430 %
6431 \def\UseTextSymbol#1#2{%
6432 %     \let\@curr@enc\cf@encoding
6433 %     \@use@text@encoding{#1}%
6434     #2%
6435 %     \@use@text@encoding\@curr@enc
6436 }
6437 \def\UseTextAccent#1#2#3{%
6438 %     \let\@curr@enc\cf@encoding
6439 %     \@use@text@encoding{#1}%
6440 %     #2{\@use@text@encoding\@curr@enc\selectfont#3}%
6441 %     \@use@text@encoding\@curr@enc
6442 }
6443 \def\@use@text@encoding#1{%
6444 %     \edef\font@encoding{#1}%
6445 %     \xdef\font@name{%
6446 %         \csname\curr@fontshape/\font@size\endcsname
6447 %     }%
6448 %     \pickup@font
6449 %     \font@name
6450 %     \@@enc@update
6451 }
6452 \def\DeclareTextSymbolDefault#1#2{%
6453     \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
6454 }
6455 \def\DeclareTextAccentDefault#1#2{%
6456     \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
6457 }
6458 \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.

```

6459 \DeclareTextAccent{"}{OT1}{127}
6460 \DeclareTextAccent{'}{OT1}{19}
6461 \DeclareTextAccent{^}{OT1}{94}
6462 \DeclareTextAccent`}{OT1}{18}
6463 \DeclareTextAccent{~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for `PLAIN TEX`.

```

6464 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
6465 \DeclareTextSymbol{\textquotedblright}{OT1}{`\'}
6466 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
6467 \DeclareTextSymbol{\textquoteright}{OT1}{``'}
6468 \DeclareTextSymbol{\i}{OT1}{16}
6469 \DeclareTextSymbol{\ss}{OT1}{25}

```

For a couple of languages we need the  $\LaTeX$ -control sequence `\scriptsize` to be available. Because plain  $\TeX$  doesn't have such a sophisticated font mechanism as  $\LaTeX$  has, we just `\let` it to `\sevenrm`.

```
6470 \ifx\scriptsize\@undefined
6471   \let\scriptsize\sevenrm
6472 \fi
6473 % End of code for plain
6474 <</Emulate LaTeX>>
```

A proxy file:

```
6475 <plain>
6476 \input babel.def
6477 </plain>
```

## 17 Acknowledgements

I would like to thank all who volunteered as  $\beta$ -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

## References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national  $\LaTeX$  styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The  $\TeX$ book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport,  *$\LaTeX$ , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in:  $\TeX$ hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Hubert Partl, *German  $\TeX$* , *TUGboat* 9 (1988) #1, p. 70–72.
- [10] Joachim Schrod, *International  $\LaTeX$  is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [11] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using  $\LaTeX$* , Springer, 2002, p. 301–373.
- [12] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).