

# Babel

Version 3.53  
2021/01/26

Johannes L. Braams  
Original author

Javier Bezos  
Current maintainer

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 . . . . .	8
1.4	Modifiers . . . . .	8
1.5	Troubleshooting . . . . .	9
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 . . . . .	16
1.12	The base option . . . . .	18
1.13	ini files . . . . .	19
1.14	Selecting fonts . . . . .	27
1.15	Modifying a language . . . . .	29
1.16	Creating a language . . . . .	30
1.17	Digits and counters . . . . .	34
1.18	Dates . . . . .	35
1.19	Accessing language info . . . . .	35
1.20	Hyphenation and line breaking . . . . .	37
1.21	Selection based on BCP 47 tags . . . . .	39
1.22	Selecting scripts . . . . .	40
1.23	Selecting directions . . . . .	41
1.24	Language attributes . . . . .	45
1.25	Hooks . . . . .	46
1.26	Languages supported by babel with ldf files . . . . .	47
1.27	Unicode character properties in luatex . . . . .	48
1.28	Tweaking some features . . . . .	49
1.29	Tips, workarounds, known issues and notes . . . . .	49
1.30	Current and future work . . . . .	50
1.31	Tentative and experimental code . . . . .	50
<b>2</b>	<b>Loading languages with language.dat</b>	<b>51</b>
2.1	Format . . . . .	51
<b>3</b>	<b>The interface between the core of babel and the language definition files</b>	<b>52</b>
3.1	Guidelines for contributed languages . . . . .	53
3.2	Basic macros . . . . .	54
3.3	Skeleton . . . . .	55
3.4	Support for active characters . . . . .	56
3.5	Support for saving macro definitions . . . . .	56
3.6	Support for extending macros . . . . .	57
3.7	Macros common to a number of languages . . . . .	57
3.8	Encoding-dependent strings . . . . .	57
<b>4</b>	<b>Changes</b>	<b>61</b>
4.1	Changes in babel version 3.9 . . . . .	61
<b>II</b>	<b>Source code</b>	<b>61</b>

<b>5</b>	<b>Identification and loading of required files</b>	<b>62</b>
<b>6</b>	<b>locale directory</b>	<b>62</b>
<b>7</b>	<b>Tools</b>	<b>62</b>
7.1	Multiple languages . . . . .	67
7.2	The Package File (L <sup>A</sup> T <sub>E</sub> X, babel.sty) . . . . .	67
7.3	base . . . . .	69
7.4	Conditional loading of shorthands . . . . .	71
7.5	Cross referencing macros . . . . .	72
7.6	Marks . . . . .	75
7.7	Preventing clashes with other packages . . . . .	76
7.7.1	ifthen . . . . .	76
7.7.2	varioref . . . . .	77
7.7.3	hhline . . . . .	77
7.7.4	hyperref . . . . .	77
7.7.5	fancyhdr . . . . .	78
7.8	Encoding and fonts . . . . .	78
7.9	Basic bidi support . . . . .	80
7.10	Local Language Configuration . . . . .	85
<b>8</b>	<b>The kernel of Babel (babel.def, common)</b>	<b>89</b>
8.1	Tools . . . . .	89
<b>9</b>	<b>Multiple languages</b>	<b>90</b>
9.1	Selecting the language . . . . .	93
9.2	Errors . . . . .	101
9.3	Hooks . . . . .	103
9.4	Setting up language files . . . . .	105
9.5	Shorthands . . . . .	107
9.6	Language attributes . . . . .	117
9.7	Support for saving macro definitions . . . . .	118
9.8	Short tags . . . . .	119
9.9	Hyphens . . . . .	120
9.10	Multiencoding strings . . . . .	121
9.11	Macros common to a number of languages . . . . .	128
9.12	Making glyphs available . . . . .	128
9.12.1	Quotation marks . . . . .	128
9.12.2	Letters . . . . .	130
9.12.3	Shorthands for quotation marks . . . . .	131
9.12.4	Umlauts and tremas . . . . .	132
9.13	Layout . . . . .	133
9.14	Load engine specific macros . . . . .	133
9.15	Creating and modifying languages . . . . .	134
<b>10</b>	<b>Adjusting the Babel bahavior</b>	<b>153</b>
<b>11</b>	<b>Loading hyphenation patterns</b>	<b>155</b>
<b>12</b>	<b>Font handling with fontspec</b>	<b>160</b>

<b>13</b>	<b>Hooks for XeTeX and LuaTeX</b>	<b>164</b>
13.1	XeTeX . . . . .	164
13.2	Layout . . . . .	166
13.3	LuaTeX . . . . .	168
13.4	Southeast Asian scripts . . . . .	173
13.5	CJK line breaking . . . . .	177
13.6	Automatic fonts and ids switching . . . . .	177
13.7	Layout . . . . .	188
13.8	Auto bidi with basic and basic-r . . . . .	191
<b>14</b>	<b>Data for CJK</b>	<b>201</b>
<b>15</b>	<b>The ‘nil’ language</b>	<b>202</b>
<b>16</b>	<b>Support for Plain T<sub>E</sub>X (plain.def)</b>	<b>202</b>
16.1	Not renaming hyphen.tex . . . . .	202
16.2	Emulating some L <sub>A</sub> T <sub>E</sub> X features . . . . .	203
16.3	General tools . . . . .	204
16.4	Encoding related macros . . . . .	207
<b>17</b>	<b>Acknowledgements</b>	<b>210</b>

## Troubleshooting

Paragraph ended before \UTFviii@three@octets was complete . . . . .	5
No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the format . . . . .	6
You are loading directly a language style . . . . .	9
Unknown language ‘LANG’ . . . . .	9
Argument of \language@active@arg” has an extra } . . . . .	13
Package fontspec Warning: ‘Language ‘LANG’ not available for font ‘FONT’ with script ‘SCRIPT’ ‘Default’ language used instead’ . . . . .	28
Package babel Info: The following fonts are not babel standard families . . . . .	29

# Part I

## User guide

**What is this document about?** This user guide focuses on internationalization and localization with  $\LaTeX$  and `pdftex`, `xetex` and `luatex` with the `babel` package. There are also some notes on its use with Plain  $\TeX$ . Part II describes the code, and usually it can be ignored.

**What if I'm interested only in the latest changes?** 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 can be still unstable.

**Can I help?** Sure! If you are interested in the  $\TeX$  multilingual support, please join the [kadingira mail list](#). You can follow the development of `babel` in [GitHub](#) and make suggestions; feel free to fork it and make pull requests. If you are the author of a package, send to me a few test files which I'll add to mine, so that possible issues can be caught in the development phase.

**It doesn't work for me!** You can ask for help in some forums like [tex.stackexchange](#), but if you have found a bug, I strongly beg you to report it in [GitHub](#), which is much better than just complaining on an e-mail list or a web forum. Remember *warnings are not errors* by themselves, they just warn about possible problems or incompatibilities.

**How can I contribute a new language?** See section 3.1 for contributing a language.

**I only need learn the most basic features.** The first subsections (1.1-1.3) describe the traditional way of loading a language (with `ldf` files), which is usually all you need. The alternative way based on `ini` files, which complements the previous one (it does *not* replace it, although it is still necessary in some languages), is described below; go to 1.13.

**I don't like manuals. I prefer sample files.** This manual contains lots of examples and tips, but in [GitHub](#) there are many [sample files](#).

## 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. Another approach is making the language a global option in order to let other packages detect and use it. This is the standard way in  $\LaTeX$  for an option – in this case a language – to be recognized by several packages.

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. It assumes UTF-8, the default encoding:

PDFTEX

```
\documentclass{article}

\usepackage[T1]{fontenc}

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

Now consider something like:

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

With this setting, the package `varioref` will also see the option `french` and will be able to use it.

**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[russian]{article}

\usepackage{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 can 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.

**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.

**NOTE** Although it has been customary to recommend placing `\title`, `\author` and other elements printed by `\maketitle` after `\begin{document}`, mainly because of shorthands, it is advisable to keep them in the preamble. Currently there is no real need to use shorthands in those macros.

## 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 can 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 with pdfTeX follows. The main language is french, which is activated when the document begins. It assumes UTF-8:

PDFTEX

```
\documentclass{article}  
  
\usepackage[T1]{fontenc}  
  
\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}
```

**NOTE** Once loaded a language, you can select it with the corresponding BCP47 tag. See section 1.21 for further details.

### 1.3 Mostly monolingual documents

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

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

**EXAMPLE** A trivial document with the default font in English and Spanish, and FreeSerif in Russian is:

LUATEX/XETEX

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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

**NOTE** Instead of its name, you may prefer to select the language with the corresponding BCP47 tag. This alternative, however, must be activated explicitly, because a two- or tree-letter word is a valid name for a language (eg, `yi`). See section 1.21 for further details.

### 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.

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

## 1.5 Troubleshooting

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

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

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

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

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

## 1.6 Plain

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

```
\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` {⟨language⟩}

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:

<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”.

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

```
\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 `bidi` 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 `bidi` 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’, 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 deprecated and `otherlanguage*` (the starred version) is preferred, because 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` {*<tag1>* = *<language1>*, *<tag2>* = *<language2>*, ...}

**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.

**WARNING** There is a clear drawback to this feature, namely, the ‘prefix’ `\text...` is heavily overloaded in  $\TeX$  and conflicts with existing macros may arise (`\textlatin`, `\textbar`, `\textit`, `\textcolor` and many others). The same applies to environments, because arabic conflicts with `\arabic`. Except if there is a reason for this ‘syntactical sugar’, the best option is to stick to the default selectors or to define your own alternatives.

**EXAMPLE** With

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

you can write

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

and

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

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

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

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

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

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

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

```
\babelensure{polish}
```

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

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

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

## 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 `\knbcode`, and `luatex` can manipulate the glyph list. Tools for point 3 can be still very useful in general. There are four levels of shorthands: *user*, *language*, *system*, and *language user* (by order of precedence). In most cases, you will use only shorthands provided by languages.

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

**NOTE** Keep in mind 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, 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 deactivated with, eg, \string).

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

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

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

**\shorthandon** {<shorthands-list>}  
**\shorthandoff** \*{<shorthands-list>}

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

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

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

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

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

**\usesshorthands** \*{<char>}

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

**New 3.9a** User shorthands are not always alive, as they may be deactivated by languages (for example, if you use " for your user shorthands and switch from german to french, they stop working). Therefore, a starred version \usesshorthands\*{<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 \usesshorthands. This restriction will be lifted in a future release.

**\defineshorthand** [ $\langle\textit{language}\rangle$ ,  $\langle\textit{language}\rangle$ , ...]{ $\langle\textit{shorthand}\rangle$ }{ $\langle\textit{code}\rangle$ }

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

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

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

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

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

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

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

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

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

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

**\languageshorthands** { $\langle\textit{language}\rangle$ }

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

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

(You may also need to activate them 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:

---

<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.

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

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

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

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

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

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

**Languages with no shorthands** Croatian, English (any variety), Indonesian, Hebrew, Interlingua, Irish, Lower Sorbian, Malaysian, North Sami, Romanian, Scottish, Welsh  
**Languages with only " as defined shorthand character** Albanian, Bulgarian, Danish, Dutch, Finnish, German (old and new orthography, also Austrian), Icelandic, Italian, Norwegian, Polish, Portuguese (also Brazilian), Russian, Serbian (with Latin script), Slovene, Swedish, Ukrainian, Upper Sorbian

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

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

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

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

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

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

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



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

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

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

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

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

## 1.11 Package options

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

**KeepShorthandsActive** Tells babel not to deactivate shorthands after loading a language file, so that they are also available in the preamble.

**activeacute** For some languages babel supports this options to set `'` as a shorthand in case it is not done by default.

**activegrave** Same for ```.

**shorthands=** `<char><char>... | off`

The only language shorthands activated are those given, like, eg:

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

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

**safe=** `none | ref | bib`

Some  $\TeX$  macros are redefined so that using shorthands is safe. With `safe=bib` only `\nocite`, `\bibcite` and `\bibitem` are redefined. With `safe=ref` only `\newlabel`, `\ref` and `\pageref` are redefined (as well as a few macros from `varioref` and `ifthen`).

With `safe=none` no macro is redefined. This option is strongly recommended, because a good deal of incompatibilities and errors are related to these redefinitions. As of **New 3.34**, in  $\epsilon$ T<sub>E</sub>X based engines (ie, almost every engine except the oldest ones) shorthands can be used in these macros (formerly you could not).

- math=** active | normal
- Shorthands are mainly intended for text, not for math. By setting this option with the value `normal` they are deactivated in math mode (default is `active`) and things like  $\{a'\}$  (a closing brace after a shorthand) are not a source of trouble anymore.
- config=**  $\langle file \rangle$
- Load  $\langle file \rangle$ .`cfg` instead of the default config file `bblopts.cfg` (the file is loaded even with `noconfigs`).
- main=**  $\langle language \rangle$
- Sets the main language, as explained above, ie, this language is always loaded last. If it is not given as package or global option, it is added to the list of requested languages.
- headfoot=**  $\langle language \rangle$
- By default, headlines and footlines are not touched (only marks), and if they contain language-dependent macros (which is not usual) there may be unexpected results. With this option you may set the language in heads and foots.
- noconfigs** Global and language default config files are not loaded, so you can make sure your document is not spoilt by an unexpected `.cfg` file. However, if the key `config` is set, this file is loaded.
- showlanguages** Prints to the log the list of languages loaded when the format was created: number (remember dialects can share it), name, hyphenation file and exceptions file.
- nocase** **New 3.9l** Language settings for uppercase and lowercase mapping (as set by `\SetCase`) are ignored. Use only if there are incompatibilities with other packages.
- silent** **New 3.9l** No warnings and no *infos* are written to the log file.<sup>9</sup>
- strings=** generic | unicode | encoded |  $\langle label \rangle$  |  $\langle font encoding \rangle$
- Selects the encoding of strings in languages supporting this feature. Predefined labels are `generic` (for traditional 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;

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

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

**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. 1.23.

**layout=**

**New 3.16** Selects which layout elements are adapted in bidi documents. See sec. 1.23.

## 1.12 The base option

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

**\AfterBabelLanguage** `{<option-name>}{<code>}`

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

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

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

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

```
\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.

<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.

### 1.13 ini files

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

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

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}
```

**New 3.49** Alternatively, you can tell babel to load all or some languages passed as options with \babelprovide and not from the ldf file in a few typical cases. Thus, provide=\* means 'load the main language with the \babelprovide mechanism instead of the ldf file' applying the basic features, which in this case means import, main. There are (currently) three options:

- provide=\* is the option just explained, for the main language;
- provide+=\* is the same for additional languages (the main language is still the ldf file);
- provide\*=\* is the same for all languages, ie, main and additional.

**EXAMPLE** The preamble in the previous example can be more compactly written as:

```
\documentclass{book}
\usepackage[georgian, provide=*]{babel}
\babelfont{rm}[Renderer=Harfbuzz]{DejaVu Sans}
```

Or also:

```
\documentclass[georgian]{book}
\usepackage[provide=*]{babel}
\babelfont{rm}[Renderer=Harfbuzz]{DeJaVu Sans}
```

**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). The Harfbuzz renderer has still some issues, so as a rule of thumb prefer the default renderer, and resort to Harfbuzz only if the former does not work for you. Fortunately, fonts can be loaded twice with different renderers; for example:

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

**Arabic** Monolingual documents mostly work in luatex, but it must be fine tuned, particularly graphical elements like `picture`. In xetex babel resorts to the  `bidi` package, which seems to work.

**Hebrew** Niqqud marks seem to work in both engines, but depending on the font cantillation marks might be 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 `Renderer=Harfbuzz`. They also work with xetex, although unlike with luatex fine tuning the font behavior 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. Khmer clusters are rendered wrongly with the default renderer. The comment about Indic scripts and lualatex also applies here. Some quick patterns can help, with something similar to:

```
\babelprovide[import,hyphenrules=+]{lao}
\babelpatterns[lao]{ໂ໋ ລ໋ ລ໌ ລ໎ ລ໑ ລ໒} % Random
```

**East Asia scripts** Settings for either Simplified or Traditional should work out of the box, with basic line breaking with any renderer. 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[japanese]{ltjbook}
\usepackage{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). With xetex both combining characters and hyphenation work as expected (not quite, but in most cases it works; the problem here are font clusters).

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

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>1</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>1</sup>
ia	Interlingua <sup>ul</sup>	ms-SG	Malay <sup>1</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
kln	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>
ksh	Colognian	pt	Portuguese <sup>ul</sup>
kw	Cornish	qu	Quechua
ky	Kyrgyz	rm	Romansh <sup>ul</sup>
lag	Langi	rn	Rundi
lb	Luxembourgish	ro	Romanian <sup>ul</sup>
lg	Ganda	rof	Rombo
lkt	Lakota	ru	Russian <sup>ul</sup>
ln	Lingala	rw	Kinyarwanda
lo	Lao <sup>ul</sup>	rwk	Rwa
lrc	Northern Luri	sa-Beng	Sanskrit
		sa-Deva	Sanskrit

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

---

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	arabic-morocco
akan	arabic-MA
albanian	arabic-syria
american	arabic-SY
amharic	armenian
ancientgreek	assamese
arabic	asturian
arabic-algeria	asu
arabic-DZ	australian



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
ben	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
chinese-simplified-singapore	german-at
chinese-simplified	german-austria
chinese-traditional-hongkongsarchina	german-ch
chinese-traditional-macausarchina	german-switzerland
chinese-traditional	german
chinese	greek
churchslavic	gujarati
churchslavic-cyrs	gusii
churchslavic-oldcyrillic <sup>13</sup>	hausa-gh
churchsslavic-glag	hausa-ghana

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

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
luxembourgish	punjabi-arabic
luyia	punjabi-gurmukhi
macedonian	punjabi-guru
machame	punjabi
makhuwameetto	quechua
makonde	romanian
malagasy	romansh
malay-bn	rombo
malay-brunei	rundi
malay-sg	russian

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

**NOTE** See the note in the previous section about some issues in specific languages.

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}
```

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

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

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 can be problematic, and also preserving 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 can 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. In the case of caption names a specific macro is provided, because this is perhaps the most frequent change:

`\setlocalecaption`  $\{\langle language-name \rangle\}\{\langle caption-name \rangle\}\{\langle string \rangle\}$

**New 3.51** Here *caption-name* is the name as string without the trailing name. An example, which also shows caption names are often a stylistic choice, is:

```
\setlocalecaption{english}{contents}{Table of Contents}
```

This works not only with existing caption names, because it also serves to define new ones by setting the *caption-name* to the name of your choice (name will be postpended). Captions so defined or redefined behave with the ‘new way’ described in the following note.

**NOTE** There are a few alternative methods:

- 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.)

- 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. This redefinition is not activated until the language is selected.

- 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}
```

This redefinition is immediate.

**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.

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

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

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

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

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. Without the optional argument it just loads some additional tools if provided by the `ini` file, like extra counters.

## 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 it
(babel)                after the language has been loaded (typically
(babel)                in the preamble) with the help of
(babel)                \setlocalecaption. An example is:
(babel)                \setlocalecaption{mylang}{chapter}{...}
(babel)                Reported on input line 18.

```

In most cases, you will only need to define a few macros. Note languages loaded on the fly are not yet available in the preamble.

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

```

\usepackage[danish]{babel}
\babelprovide{arhinish}
\setlocalecaption{arhinish}{chapter}{Chapitula}
\setlocalecaption{arhinish}{refname}{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 and date (also line breaking rules in newly defined languages). 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 can be written:

```

\babelprovide[import]{hungarian}

```

There are about 250 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 may show a warning about the current lack of suitability of some features.

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=**  $\langle\text{language-tag}\rangle$

Loads only the strings. For example:

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

**hyphenrules=**  $\langle\text{language-list}\rangle$

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

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

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

A special value is +, which allocates a new language (in the  $\text{\TeX}$  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 can try:

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

Remember there is an alternative syntax for the latter:

```
\usepackage[italian, polytonicgreek, provide=*]{babel}
```

**script=**  $\langle\text{script-name}\rangle$

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

**language=**  $\langle\text{language-name}\rangle$

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

**alph=**  $\langle\text{counter-name}\rangle$

Assigns to `\alph` that counter. See the next section.

**Alph=**  $\langle\text{counter-name}\rangle$

Same for `\Alph`.

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 (as its name implies, it acts on characters, not on spaces). 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 or modified with `\babelcharproperty`.

**NOTE** An alternative approach with luatex and Harfbuzz is the font option `RawFeature={multiscript=auto}`. It does not switch the babel language and therefore the line breaking rules, but in many cases it can be enough.

**intraspace=**  $\langle\text{base}\rangle$   $\langle\text{shrink}\rangle$   $\langle\text{stretch}\rangle$

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=**  $\langle\text{penalty}\rangle$

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).

**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.

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

## 1.17 Digits and counters

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

For example:

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

Languages providing native digits in all or some variants are:

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

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

**NOTE** With xetex you can use the option `Mapping` when defining a font.

**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 unprotected `\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):

- `\localnumeral{<style>}{<number>}`, like `\localnumeral{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  
**Amharic** afar, agaw, ari, blin, dizi, gedeo, gumuz, hadiyya, harari, kaffa, kebena, kembata, konso, kunama, meen, oromo, saho, sidama, silti, tigre, wolaita, yemsa  
**Arabic** abjad, maghrebi.abjad  
**Belarusan, Bulgarian, Macedonian, Serbian** lower, upper  
**Bengali** alphabetic  
**Coptic** epact, lower.letters  
**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, syllable, hanja.informal, hanja.formal, hangul.formal, cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha  
**Marathi** alphabetic  
**Persian** abjad, alphabetic  
**Russian** lower, lower.full, upper, upper.full  
**Syriac** letters  
**Tamil** ancient  
**Thai** alphabetic  
**Ukrainian** lower, lower.full, upper, upper.full  
**Chinese** cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha

**New 3.45** In addition, native digits (in languages defining them) may be printed with the numeral style digits.

## 1.18 Dates

**New 3.45** When the data is taken from an ini file, you may print the date corresponding to the Gregorian calendar and other lunisolar systems with the following command.

**\localedate** [*<calendar=.., variant=..>*]{*<year>*}{*<month>*}{*<day>*}

By default the calendar is the Gregorian, but a ini files may define strings for other calendars (currently ar, ar-\*, he, fa, hi.) In the latter case, the three arguments are the year, the month, and the day in those in the corresponding calendar. They are *not* the Gregorian data to be converted (which means, say, 13 is a valid month number with calendar=hebrew).

Even with a certain calendar there may be variants. In Kurmanji the default variant prints something like 30. *Çileyê Pêşîn 2019*, but with variant=iza fa it prints 31'ê *Çileyê Pêşînê 2019*.

## 1.19 Accessing language info

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

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

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

If more than one language is used, it might be necessary to know which language is active at a specific time. This can be checked by a call to `\iflanguage`, but note here “language” is used in the  $\TeX$ 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 macro 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 full BCP 47 tag (see the warning below).

`language.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 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).

**WARNING** **New 3.46** As of version 3.46 `tag.bcp47` returns the full BCP 47 tag. Formerly it returned just the language subtag, which was clearly counterintuitive.

**\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 פֶּרֶק.

If the key does not exist, the macro is set to `\relax` and an error is raised. **New 3.47** With the starred version no error is raised, so that you can take your own actions with undefined properties.

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 patterns (which, in turn, is just a component of the line breaking algorithm described in the next section). The data about preloaded patterns are stored 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.

## 1.20 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` `*{<type>}`  
`\babelhyphen` `*{<text>}`

**New 3.9a** It is customary to classify hyphens in two types: (1) *explicit* or *hard hyphens*, which in T<sub>E</sub>X are entered as -, and (2) *optional* or *soft hyphens*, which are entered as \-. Strictly, a *soft hyphen* is not a hyphen, but just a breaking opportunity or, in T<sub>E</sub>X terms, a “discretionary”; a *hard hyphen* is a hyphen with a breaking opportunity after it. A further type is a *non-breaking hyphen*, a hyphen without a breaking opportunity. In T<sub>E</sub>X, - and \- forbid further breaking opportunities in the word. This is the desired behavior very often, but not always, and therefore many languages provide shorthands for these cases. Unfortunately, this has not been done consistently: for example, - in Dutch, Portuguese, Catalan or Danish is a hard hyphen, while in German, Spanish, Norwegian, Slovak or Russian is a soft hyphen. Furthermore, some of them even redefine \-, so that you cannot insert a soft hyphen without breaking opportunities in the rest of the word. Therefore, some macros are provided with a set of basic “hyphens” which can be used by themselves, to define a user shorthand, or even in language files.

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

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

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

There are also some differences with L<sup>A</sup>T<sub>E</sub>X: (1) the character used is that set for the current font, while in L<sup>A</sup>T<sub>E</sub>X it is hardwired to - (a typical value); (2) the hyphen to be used in fonts with a negative `\hyphenchar` is -, like in L<sup>A</sup>T<sub>E</sub>X, but it can be changed to another value by redefining `\babelnullhyphen`; (3) a break after the hyphen is forbidden if preceded by a glue >0 pt (at the beginning of a word, provided it is not immediately preceded by, say, a parenthesis).

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

**New 3.9a** Sets hyphenation exceptions for the languages given or, without the optional argument, for *all* languages (eg, proper nouns or common loan words, and of course monolingual documents). Language exceptions take precedence over global ones. It can be used only in the preamble, and exceptions are set when the language is first selected, thus taking into account changes of `\lccodes`’s done in `\extras{lang}` as well as the language-specific encoding (not set in the preamble by default). Multiple `\babelhyphenation`’s are allowed. For example:

```
\babelhyphenation{Wal-hal-la Dar-bhan-ga}
```

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

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

**\babelpatterns** [*<language>* , <language> , ... ] { <patterns> }

**New 3.9m** In *luatex* only,<sup>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 `([îû])`, the replacement could be `{1|îû|íú}`, which maps `î` to `í`, and `û` to `ú`, so that the diaeresis is removed.

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

<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.

See the [babel wiki](#) for a more detailed description and some examples. It also describes a few additional replacement types (string, penalty).

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).

You are limited to substitutions as done by lua, although a future implementation may alternatively accept lpeg.

**`\babelprehyphenation`** `{\langle locale-name \rangle}{\langle lua-pattern \rangle}{\langle replacement \rangle}`

**New 3.44-3-52** This command is not strictly about hyphenation, but it is included here because it is a clear counterpart of `\babelposthyphenation`. It is similar to the latter, 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, while `|` stands for an ordinary space; (3) in the replacement, discretionaries are not accepted.

It handles glyphs and spaces (but you can not insert spaces).

Performance is still somewhat poor in some cases, but it is fast in the typical ones.

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

**EXAMPLE** You can replace a character (or series of them) by another character (or series of them). Thus, to enter *ž* as *zh* and *š* as *sh* in a newly created locale for transliterated Russian:

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

**EXAMPLE** The following rule prevent the word “a” from being at the end of a line:

```
\babelprehyphenation{english}{|a|}
{ }, { }, % Keep first space and a
{insert, penalty = 10000}, % Insert penalty
{ } % Keep last space
}
```

## 1.21 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`  $\rightarrow$  `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{
  autoload.bcp47 = on,
  autoload.bcp47.options = import
}

\begin{document}

Chapter in Danish: \chaptername.

\selectlanguage{de-AT}

\localedate{2020}{1}{30}

\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:

`autoload.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.

**New 3.46** If an `ldf` file has been loaded, you can enable the corresponding language tags as selector names with:

```
\babeladjust{ bcp47.toname = on }
```

(You can deactivate it with `off`.) So, if `dutch` is one of the package (or class) options, you can write `\selectlanguage{nl}`. Note the language name does not change (in this example is still `dutch`), but you can get it with `\localeinfo` or `\getlanguageproperty`. It must be turned on explicitly for similar reasons to those explained above.

## 1.22 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.

## 1.23 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 can 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 can be improvements in the future, because setting bidi text has many subtleties (see for example <https://www.w3.org/TR/html-bidi/>). A basic stable version for other engines must wait. This applies to text; there is a basic support for **graphical** elements, including the `picture` environment (with `pict2e`) and `pfg/tikz`. Also, indexes and the like are under study, as well as math (there is progress in the latter, too, but for example cases may fail).

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

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

---

<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.

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}
```

```

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

\begin{document}

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

\end{document}

```

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

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

```

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

```

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

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

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

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

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

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

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

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

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

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

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

**tabular** required in `luatex` for R `tabular`, so that the first column is the right one (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 (**With recent versions of L<sup>A</sup>T<sub>E</sub>X, this feature has stopped working**). 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:

```
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** `{\langle section-name \rangle}`

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

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

**New 3.17** Something like:

```
\BabelFootnote{\parsfootnote}{\language name}{\{ \}}
```

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

```
\footnote{(\foreignlanguage{\language name}{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 name}{\{ \}}%
\BabelFootnote{\localfootnote}{\language name}{\{ \}}%
\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.

## 1.24 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).

## 1.25 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  $\text{T}_{\text{E}}\text{X}$  parameters (`#1`, `#2`, `#3`), with the meaning given:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## 1.26 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>

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



**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  $\LaTeX$ .

## 1.27 Unicode character properties in luatex

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

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

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

```

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

```

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

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

## 1.28 Tweaking some features

`\babeladjust` `{<key-value-list>}`

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

## 1.29 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`),  $\TeX$  will keep complaining about an undefined label. To prevent such problems, you can 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}}
```

- For the hyphenation to work correctly, `lccodes` cannot change, because  $\TeX$  only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.<sup>20</sup> So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of  $\TeX$ , not of `babel`. Alternatively, you may use `\usesorthands` to activate `'` and `\defineshortand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).
- `\bibitem` is out of sync with `\selectlanguage` in the `.aux` file. The reason is `\bibitem` uses `\immediate` (and others, in fact), while `\selectlanguage` doesn't. There is no known workaround.

<sup>20</sup>This explains why  $\TeX$  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.

- 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.

### 1.30 Current and future work

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

### 1.31 Tentative and experimental code

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

#### Options for locales loaded on the fly

**New 3.51** `\babeladjust{ autoload.options = ... }` sets the options when a language is loaded on the fly (by default, no options). A typical value would be `import`, which

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

defines captions, date, numerals, etc., but ignores the code in the tex file (for example, extended numerals in Greek).

### Labels

**New 3.48** There is some work in progress for babel to deal with labels, both with the relation to captions (chapters, part), and how counters are used to define them. It is still somewhat tentative because it is far from trivial – see the wiki for further details.

## 2 Loading languages with language.dat

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

**New 3.9q** With luatex, however, patterns are loaded on the fly when requested by the language (except the “0th” language, typically english, which is preloaded always).<sup>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 T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X that the hyphenation patterns just processed have to be known under an alternative name. Here is an example:

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

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

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

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

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

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

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

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

With the previous settings, if the encoding when the language is selected is T1 then the patterns in `hyphenT1.ger` are used, but otherwise use those in `hyphen.ger` (note the encoding can be set in `\extras⟨lang⟩`).  
A typical error when using babel is the following:

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

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

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

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

The following assumptions are made:

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

Some recommendations:

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

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

### 3.1 Guidelines for contributed languages

Currently, the easiest way to contribute a new language is by taking one of the 500 or so `ini` templates available on GitHub as a basis. Just make a pull request or download it and then, after filling the fields, send it to me. Feel free to ask for help or to make feature requests.

As to `ldf` files, now language files are “outsourced” and are located in a separate directory (`/macros/latex/contrib/babel-contrib`), so that they are contributed directly to CTAN (please, do not send to me language styles just to upload them to CTAN).

Of course, placing your style files in this directory is not mandatory, but if you want to do it, here are a few guidelines.

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

---

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

The following page provides a starting point for ldf files:

<http://www.texnia.com/incubator.html>. See also

<https://github.com/latex3/babel/wiki/List-of-locale-templates>.

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 L<sup>A</sup>T<sub>E</sub>X command `\ProvidesPackage`.

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



<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, $\text{\LaTeX}$ 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 $\text{\LaTeX}$ 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\monthiname{<name of first month>}
% More strings

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

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

\EndBabelCommands

```



```

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

\ldf@finish{<language>}

```

**NOTE** If for some reason you want to load a package in your style, you should be aware it cannot be done directly in the ldf file, but it can be delayed with `\AtEndOfPackage`. Macros from external packages can be used *inside* definitions in the ldf itself (for example, `\extras<language>`), but if executed directly, the code must be placed inside `\AtEndOfPackage`. A trivial example illustrating these points is:

```

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

```

### 3.4 Support for active characters

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

`\initiate@active@char`

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

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

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

`\declare@shorthand`

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

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

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

### 3.5 Support for saving macro definitions

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

`\babel@save`

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

`\babel@savevariable`

A second macro is provided to save the current value of a variable. In this context,

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

anything that is allowed after the `\the` primitive is considered to be a variable. The macro takes one argument, the *<variable>*.

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

### 3.6 Support for extending macros

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

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

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

```

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

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

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

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

\EndBabelCommands

```

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

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

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

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

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

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

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

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

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

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

**\SetStringLoop** {<macro-name>}{<string-list>}

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

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

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

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

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

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

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

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

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

## Part II

# Source code

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

## 5 Identification and loading of required files

*Code documentation is still under revision.*

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

The babel package after unpacking consists of the following files:

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

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

**babel.sty** is the  $\LaTeX$  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 an uppercase letter. It can be just a letter (eg, babel.name.A, babel.name.B) or a name (eg, date.long.Nominative, date.long.Formal, but no language is currently using the latter). *Multi-letter* qualifiers are forward compatible in the sense they won’t conflict with new “global” keys (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=3.53>>
2 <<date=2021/01/26>>
```

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

We define some basic macros which just make the code cleaner. `\bbl@add` is now used internally instead of `\addto` because of the unpredictable behavior of the latter. Used in `babel.def` and in `babel.sty`, which means in  $\LaTeX$  is executed twice, but we need them when defining options and

babel.def cannot be load until options have been defined. This does not hurt, but should be fixed somehow.

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

**\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

```

<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.



```

40 \expandafter\bb1@trim@b
41 \else
42 \expandafter\bb1@trim@b\expandafter#1%
43 \fi}%
44 \long\def\bb1@trim@b#1##1 \nil{\bb1@trim@i##1}}
45 \bb1@tempa{ }
46 \long\def\bb1@trim@i#1\@nil#2\relax#3{#3{#1}}
47 \long\def\bb1@trim@def#1{\bb1@trim{\def#1}}

```

**\bb1@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\bb1@ifunset#1{%
50 \expandafter\ifx\csname#1\endcsname\relax
51 \expandafter\@firstoftwo
52 \else
53 \expandafter\@secondoftwo
54 \fi}
55 \bb1@ifunset{ifcsname}%
56 {}%
57 {\gdef\bb1@ifunset#1{%
58 \ifcsname#1\endcsname
59 \expandafter\ifx\csname#1\endcsname\relax
60 \bb1@afterelse\expandafter\@firstoftwo
61 \else
62 \bb1@afterfi\expandafter\@secondoftwo
63 \fi
64 \else
65 \expandafter\@firstoftwo
66 \fi}}
67 \endgroup

```

**\bb1@ifblank** A tool from url, by Donald Arseneau, which tests if a string is empty or space. The companion macros tests if a macro is defined with some ‘real’ value, ie, not `\relax` and not empty,

```

68 \def\bb1@ifblank#1{%
69 \bb1@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
70 \long\def\bb1@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
71 \def\bb1@ifset#1#2#3{%
72 \bb1@ifunset{#1}{#3}{\bb1@exp{\bb1@ifblank{#1}}{#3}{#2}}}

```

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).

```

73 \def\bb1@forkv#1#2{%
74 \def\bb1@kvcmd##1##2##3{#2}%
75 \bb1@kvnext#1,\@nil,}
76 \def\bb1@kvnext#1,{%
77 \ifx\@nil#1\relax\else
78 \bb1@ifblank{#1}{\bb1@forkv@eq#1=\@empty=\@nil{#1}}%
79 \expandafter\bb1@kvnext
80 \fi}
81 \def\bb1@forkv@eq#1=#2=#3\@nil#4{%
82 \bb1@trim@def\bb1@forkv@a{#1}%
83 \bb1@trim{\expandafter\bb1@kvcmd\expandafter{\bb1@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).

```

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

```

\bbl@replace

```

93 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
94   \toks@{ }%
95   \def\bbl@replace@aux##1#2##2#2{%
96     \ifx\bbl@nil##2%
97       \toks@\expandafter{\the\toks@##1}%
98     \else
99       \toks@\expandafter{\the\toks@##1#3}%
100     \bbl@afterfi
101     \bbl@replace@aux##2#2%
102   \fi}%
103   \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
104   \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).

```

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

```

Two further tools. \bbl@samestring first expand its arguments and then compare their expansion

(sanitized, so that the catcodes do not matter). `\bbl@engine` takes the following values: 0 is pdf $\TeX$ , 1 is  $\text{\LaTeX}$ , and 2 is  $\text{\XeTeX}$ . You may use the latter in your language style if you want.

```

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

```

A somewhat hackish tool (hence its name) to avoid spurious spaces in some contexts.

```

154 \def\bbl@bsphack{%
155   \ifhmode
156     \hskip\z@skip
157     \def\bbl@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
158   \else
159     \let\bbl@esphack\@empty
160   \fi}

```

Another hackish tool, to apply case changes inside a protected macros. It's based on the internal `\let's` made by `\MakeUppercase` and `\MakeLowercase` between things like `\oe` and `\OE`.

```

161 \def\bbl@cased{%
162   \ifx\oe\OE
163     \expandafter\in@\expandafter
164       {\expandafter\OE\expandafter}\expandafter{\oe}%
165     \ifin@
166       \bbl@afterelse\expandafter\MakeUppercase
167     \else
168       \bbl@afterfi\expandafter\MakeLowercase
169     \fi
170   \else
171     \expandafter\@firstofone
172   \fi}
173 <</Basic macros>>

```

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

```

174 <<*Make sure ProvidesFile is defined>> ≡
175 \ifx\ProvidesFile\@undefined
176   \def\ProvidesFile#1[#2 #3 #4]{%
177     \wlog{File: #1 #4 #3 <#2>}%
178     \let\ProvidesFile\@undefined}
179 \fi
180 <</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.

```
181 <<*Define core switching macros>> ≡
182 <<ifx\language\undefined
183   \csname newcount\endcsname\language
184 \fi
185 <</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.

```
186 <<*Define core switching macros>> ≡
187 <<*Define core switching macros>> ≡
188 \countdef\last@language=19 % TODO. why? remove?
189 \def\addlanguage{\csname newlanguage\endcsname}
190 <</Define core switching macros>>
```

Now we make sure all required files are loaded. When the command `\AtBeginDocument` doesn't exist we assume that we are dealing with a plain-based format or L<sup>A</sup>T<sub>E</sub>X 2.09. In that case the file `plain.def` is needed (which also defines `\AtBeginDocument`, and therefore it is not loaded twice). We need the first part when the format is created, and `\orig@dump` is used as a flag. Otherwise, we need to use the second part, so `\orig@dump` is not defined (`plain.def` undefines it). Check if the current version of `switch.def` has been previously loaded (mainly, `hyphen.cfg`). If not, load it now. We cannot load `babel.def` here because we first need to declare and process the package options.

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

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

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

```
191 <*package>
192 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
193 \ProvidesPackage{babel}[\<<date>> \<<version>> The Babel package]
194 \@ifpackagewith{babel}{debug}
195   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
196    \let\bbl@debug\@firstofone
197    \ifx\directlua\@undefined\else
198      \directlua{ Babel = Babel or {}
199                Babel.debug = true }%
200    \fi}
201 {\providecommand\bbl@trace[1]{}%
202  \let\bbl@debug\@gobble
203  \ifx\directlua\@undefined\else
204    \directlua{ Babel = Babel or {}
205              Babel.debug = false }%
206  \fi}
207 <<Basic macros>>
208 % Temporarily repeat here the code for errors
209 \def\bbl@error#1#2{%
210   \begingroup
```

```

211     \def\{\MessageBreak}%
212     \PackageError{babel}{#1}{#2}%
213   \endgroup}
214 \def\bbl@warning#1{%
215   \begingroup
216     \def\{\MessageBreak}%
217     \PackageWarning{babel}{#1}%
218   \endgroup}
219 \def\bbl@infowarn#1{%
220   \begingroup
221     \def\{\MessageBreak}%
222     \GenericWarning
223       {(babel) \@spaces\@spaces\@spaces}%
224       {Package babel Info: #1}%
225   \endgroup}
226 \def\bbl@info#1{%
227   \begingroup
228     \def\{\MessageBreak}%
229     \PackageInfo{babel}{#1}%
230   \endgroup}
231   \def\bbl@nocaption{\protect\bbl@nocaption@i}
232 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
233   \global\@namedef{#2}{\textbf{?#1?}}%
234   \@nameuse{#2}%
235   \bbl@warning{% TODO.
236     \@backslashchar#2 not set. Please, define it\\%
237     after the language has been loaded (typically\\%
238     in the preamble) with the help of\\%
239     \string\setlocalecaption. An example is:\\%
240     \string\setlocalecaption{mylang}{chapter}{..}\\
241     Reported}}
242 \def\bbl@tentative{\protect\bbl@tentative@i}
243 \def\bbl@tentative@i#1{%
244   \bbl@warning{%
245     Some functions for '#1' are tentative.\\%
246     They might not work as expected and their behavior\\%
247     may change in the future.\\%
248     Reported}}
249 \def\@nolanerr#1{%
250   \bbl@error
251     {You haven't defined the language #1\space yet.\\%
252     Perhaps you misspelled it or your installation\\%
253     is not complete}%
254     {Your command will be ignored, type <return> to proceed}}
255 \def\@nopatterns#1{%
256   \bbl@warning
257     {No hyphenation patterns were preloaded for\\%
258     the language `#1' into the format.\\%
259     Please, configure your TeX system to add them and\\%
260     rebuild the format. Now I will use the patterns\\%
261     preloaded for \bbl@nulllanguage\space instead}}
262   % End of errors
263 \@ifpackagewith{babel}{silent}
264   {\let\bbl@info\@gobble
265    \let\bbl@infowarn\@gobble
266    \let\bbl@warning\@gobble}
267   {}
268 %
269 \def\AfterBabelLanguage#1{%

```

```
270 \global\expandafter\bb1@add\csname#1.ldf-h@@k\endcsname}%
```

If the format created a list of loaded languages (in `\bb1@languages`), get the name of the 0-th to show the actual language used. Also available with `base`, because it just shows info.

```
271 \ifx\bb1@languages\@undefined\else
272 \begingroup
273 \catcode`\^^I=12
274 \@ifpackagewith{babel}{showlanguages}{%
275 \begingroup
276 \def\bb1@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
277 \wlog{<*languages>}%
278 \bb1@languages
279 \wlog{</languages>}%
280 \endgroup}{%
281 \endgroup
282 \def\bb1@elt#1#2#3#4{%
283 \ifnum#2=\z@
284 \gdef\bb1@nulllanguage{#1}%
285 \def\bb1@elt##1##2##3##4{%
286 \fi}%
287 \bb1@languages
288 \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`.

```
289 \bb1@trace{Defining option 'base'}
290 \@ifpackagewith{babel}{base}{%
291 \let\bb1@onlyswitch\@empty
292 \let\bb1@provide@locale\relax
293 \input babel.def
294 \let\bb1@onlyswitch\@undefined
295 \ifx\directlua\@undefined
296 \DeclareOption*{\bb1@patterns{\CurrentOption}}%
297 \else
298 \input luababel.def
299 \DeclareOption*{\bb1@patterns@lua{\CurrentOption}}%
300 \fi
301 \DeclareOption{base}{}%
302 \DeclareOption{showlanguages}{}%
303 \ProcessOptions
304 \global\expandafter\let\csname opt@babel.sty\endcsname\relax
305 \global\expandafter\let\csname ver@babel.sty\endcsname\relax
306 \global\let\@ifl@ter@@\@ifl@ter
307 \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
308 \endinput}{}%
309 % \end{macrocode}
310 %
311 % \subsection{\texttt{key=value} options and other general option}
312 %
313 % The following macros extract language modifiers, and only real
314 % package options are kept in the option list. Modifiers are saved
315 % and assigned to |\BabelModifiers| at |\bb1@load@language|; when
316 % no modifiers have been given, the former is |\relax|. How
```

```

317%   modifiers are handled are left to language styles; they can use
318%   |\in@|, loop them with |\@for| or load |keyval|, for example.
319%
320%   \begin{macrocode}
321\bbbl@trace{key=value and another general options}
322\bbbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
323\def\bbbl@tempb#1.#2{% Remove trailing dot
324  #1\ifx\@empty#2\else,\bbbl@afterfi\bbbl@tempb#2\fi}%
325\def\bbbl@tempd#1.#2\@nnil{% TODO. Refactor lists?
326  \ifx\@empty#2%
327    \edef\bbbl@tempc{\ifx\bbbl@tempc\@empty\else\bbbl@tempc,\fi#1}%
328  \else
329    \in@{,provide,}{, #1,}%
330    \ifin@
331      \edef\bbbl@tempc{%
332        \ifx\bbbl@tempc\@empty\else\bbbl@tempc,\fi#1.\bbbl@tempb#2}%
333    \else
334      \in@{=}{ #1}%
335      \ifin@
336        \edef\bbbl@tempc{\ifx\bbbl@tempc\@empty\else\bbbl@tempc,\fi#1.#2}%
337      \else
338        \edef\bbbl@tempc{\ifx\bbbl@tempc\@empty\else\bbbl@tempc,\fi#1}%
339        \bbbl@csarg\edef{mod@#1}{\bbbl@tempb#2}%
340      \fi
341    \fi
342  \fi}
343\let\bbbl@tempc\@empty
344\bbbl@foreach\bbbl@tempa{\bbbl@tempd#1.\@empty\@nnil}
345\expandafter\let\csname opt@babel.sty\endcsname\bbbl@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.

```

346\DeclareOption{KeepShorthandsActive}{}
347\DeclareOption{activeacute}{}
348\DeclareOption{activegrave}{}
349\DeclareOption{debug}{}
350\DeclareOption{noconfigs}{}
351\DeclareOption{showlanguages}{}
352\DeclareOption{silent}{}
353\DeclareOption{mono}{}
354\DeclareOption{shorthands=off}{\bbbl@tempa shorthands=\bbbl@tempa}
355\chardef\bbbl@iniflag\z@
356\DeclareOption{provide=*}{\chardef\bbbl@iniflag\@ne} % main -> +1
357\DeclareOption{provide+=*}{\chardef\bbbl@iniflag\tw@} % add = 2
358\DeclareOption{provide*=*}{\chardef\bbbl@iniflag\thr@@} % add + main
359% A separate option
360\let\bbbl@autoload@options\@empty
361\DeclareOption{provide=@*}{\def\bbbl@autoload@options{import}}
362% Don't use. Experimental. TODO.
363\newif\ifbbbl@single
364\DeclareOption{selectors=off}{\bbbl@singletrue}
365<<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.

```

366\let\bbbl@opt@shorthands\@nnil

```

```

367 \let\bbl@opt@config\@nnil
368 \let\bbl@opt@main\@nnil
369 \let\bbl@opt@headfoot\@nnil
370 \let\bbl@opt@layout\@nnil

```

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

```

371 \def\bbl@tempa#1=#2\bbl@tempa{%
372   \bbl@csarg\ifx{opt@#1}\@nnil
373   \bbl@csarg\edef{opt@#1}{#2}%
374   \else
375   \bbl@error
376   {Bad option `#1=#2'. Either you have misspelled the\\%
377    key or there is a previous setting of `#1'. Valid\\%
378    keys are, among others, `shorthands', `main', `bidi',\\%
379    `strings', `config', `headfoot', `safe', `math'.}%
380   {See the manual for further details.}
381 \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.

```

382 \let\bbl@language@opts\@empty
383 \DeclareOption*{%
384   \bbl@xin@{\string=}{\CurrentOption}%
385   \ifin@
386   \expandafter\bbl@tempa\CurrentOption\bbl@tempa
387   \else
388   \bbl@add@list\bbl@language@opts{\CurrentOption}%
389   \fi}

```

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

```

390 \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=...

```

391 \bbl@trace{Conditional loading of shorthands}
392 \def\bbl@sh@string#1{%
393   \ifx#1\@empty\else
394     \ifx#1t\string~%
395     \else\ifx#1c\string,%
396     \else\string#1%
397     \fi\fi
398     \expandafter\bbl@sh@string
399   \fi}
400 \ifx\bbl@opt@shorthands\@nnil
401   \def\bbl@ifshorthand#1#2#3{#2}%
402 \else\ifx\bbl@opt@shorthands\@empty
403   \def\bbl@ifshorthand#1#2#3{#3}%
404 \else

```

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

```

405   \def\bbl@ifshorthand#1{%
406     \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
407     \ifin@
408     \expandafter\@firstoftwo

```



```

409 \else
410 \expandafter\@secondoftwo
411 \fi}

```

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

```

412 \edef\bbl@opt@shorthands{%
413 \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.

```

414 \bbl@ifshorthand{'}%
415 {\PassOptionsToPackage{activeacute}{babel}}{}
416 \bbl@ifshorthand{`}%
417 {\PassOptionsToPackage{activegrave}{babel}}{}
418 \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.

```

419 \ifx\bbl@opt@headfoot\@nnil\else
420 \g@addto@macro\@resetactivechars{%
421 \set@typeset@protect
422 \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
423 \let\protect\noexpand}
424 \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.

```

425 \ifx\bbl@opt@safe\@undefined
426 \def\bbl@opt@safe{BR}
427 \fi
428 \ifx\bbl@opt@main\@nnil\else
429 \edef\bbl@language@opts{%
430 \ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
431 \bbl@opt@main}
432 \fi

```

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

```

433 \bbl@trace{Defining IfBabelLayout}
434 \ifx\bbl@opt@layout\@nnil
435 \newcommand\IfBabelLayout[3]{#3}%
436 \else
437 \newcommand\IfBabelLayout[1]{%
438 \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
439 \ifin@
440 \expandafter\@firstoftwo
441 \else
442 \expandafter\@secondoftwo
443 \fi}
444 \fi

```

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

```

445 \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.

```
446 <<{*More package options}>> ≡
447 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
448 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
449 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
450 <</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.

```
451 \bbl@trace{Cross referencing macros}
452 \ifx\bbl@opt@safe\@empty\else
453   \def\@newl@bel#1#2#3{%
454     \@safe@activestrue
455     \bbl@ifunset{#1@#2}%
456       \relax
457       {\gdef\@multiplelabels{%
458         \@latex@warning@no@line{There were multiply-defined labels}}%
459         \@latex@warning@no@line{Label `#2' multiply defined}}%
460     \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.

```
461 \CheckCommand*\@testdef[3]{%
462   \def\reserved@a{#3}%
463   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
464   \else
465     \@tempswatrue
466   \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.

```
467 \def\@testdef#1#2#3{% TODO. With @samestring?
468   \@safe@activestrue
469   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
470   \def\bbl@tempb{#3}%
471   \@safe@activesfalse
472   \ifx\bbl@tempa\relax
473   \else
474     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
475   \fi
476   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
477   \ifx\bbl@tempa\bbl@tempb
478   \else
479     \@tempswatrue
480   \fi}
481 \fi
```

`\ref` `\pageref` 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.

```

482 \bbl@xin@{R}\bbl@opt@safe
483 \ifin@
484 \bbl@redefineroobust\ref#1{%
485   \@safe@activetrue\org@ref{#1}\@safe@activfalse}
486 \bbl@redefineroobust\pageref#1{%
487   \@safe@activetrue\org@pageref{#1}\@safe@activfalse}
488 \else
489   \let\org@ref\ref
490   \let\org@pageref\pageref
491 \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.

```

492 \bbl@xin@{B}\bbl@opt@safe
493 \ifin@
494 \bbl@redefine\@citex[#1]#2{%
495   \@safe@activetrue\edef\@tempa{#2}\@safe@activfalse
496   \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.

```

497 \AtBeginDocument{%
498   \@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.)

```

499   \def\@citex[#1][#2]#3{%
500     \@safe@activetrue\edef\@tempa{#3}\@safe@activfalse
501     \org@@citex[#1][#2]{\@tempa}}%
502   }{}}

```

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

```

503 \AtBeginDocument{%
504   \@ifpackageloaded{cite}{%
505     \def\@citex[#1]#2{%
506       \@safe@activetrue\org@@citex[#1]{#2}\@safe@activfalse}%
507     }{}}

```

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

```

508 \bbl@redefine\nocite#1{%
509   \@safe@activetrue\org@nocite{#1}\@safe@activfalse}

```

`\bblcite` 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 `\bblcite` is needed we define `\bblcite` in such a way that it redefines itself with the proper definition. We call `\bbl@cite@choice` to select the proper definition for `\bblcite`. This new definition is then activated.

```

510 \bbl@redefine\bblcite{%
511   \bbl@cite@choice
512   \bblcite}

```

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

```

513 \def\bbl@bibtex#1#2{%
514   \org@bibtex{#1}{\@safe@activesfalse#2}}

```

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

```

515 \def\bbl@cite@choice{%
516   \global\let\bibtex\bbl@bibtex
517   \ifpackageloaded{natbib}{\global\let\bibtex\org@bibtex}{}%
518   \ifpackageloaded{cite}{\global\let\bibtex\org@bibtex}{}%
519   \global\let\bbl@cite@choice\relax}

```

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

```

520 \AtBeginDocument{\bbl@cite@choice}

```

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

```

521 \bbl@redefine\@bibitem#1{%
522   \@safe@activestrue\org@bibitem{#1}\@safe@activesfalse}
523 \else
524   \let\org@nocite\nocite
525   \let\org@@citex\@citex
526   \let\org@bibtex\bibtex
527   \let\org@@bibitem\@bibitem
528 \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.

```

529 \bbl@trace{Marks}
530 \IfBabelLayout{sectioning}
531   {\ifx\bbl@opt@headfoot\@nnil
532     \g@addto@macro\@resetactivechars{%
533       \set@typeset@protect
534       \expandafter\select@language@x\expandafter{\bbl@main@language}%
535       \let\protect\noexpand
536       \ifcase\bbl@bidimode\else % Only with bidi. See also above
537         \edef\thepage{%
538           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
539       \fi}%
540   \fi}
541 {\ifbbl@single\else
542   \bbl@ifunset{markright }{\bbl@redefine\bbl@redefineroobust
543     \markright#1{%
544       \bbl@ifblank{#1}%
545       {\org@markright{}}}%
546     {\toks@{#1}%
547       \bbl@exp{%
548         \\org@markright{\protect\\foreignlanguage{\language}%
549           {\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,  $\text{\LaTeX}$  stores the definition in an intermediate macro, so it's not necessary anymore, but it's preserved for older versions.)

```

550     \ifx\@mkboth\markboth
551     \def\bbl@tempc{\let\@mkboth\markboth}
552     \else
553     \def\bbl@tempc{}
554     \fi
555     \bbl@ifunset{markboth }{\bbl@redefine\bbl@redefineroobust
556     \markboth#1#2{%
557     \protected@edef\bbl@tempb##1{%
558     \protect\foreignlanguage
559     {\language}\protect\bbl@restore@actives##1}}%
560     \bbl@ifblank{#1}%
561     {\toks@{}}%
562     {\toks@\expandafter{\bbl@tempb{#1}}}%
563     \bbl@ifblank{#2}%
564     {\@temptokena{}}%
565     {\@temptokena\expandafter{\bbl@tempb{#2}}}%
566     \bbl@exp{\org@markboth{\the\toks@}{\the\@temptokena}}}
567     \bbl@tempc
568     \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.

```

569 \bbl@trace{Preventing clashes with other packages}
570 \bbl@xin@{R}\bbl@opt@safe
571 \ifin@
572 \AtBeginDocument{%
573 \ifpackage{ifthen}{%
574 \bbl@redefine@long\ifthenelse#1#2#3{%
575 \let\bbl@temp@pref\pageref
576 \let\pageref\org@pageref
577 \let\bbl@temp@ref\ref
578 \let\ref\org@ref
579 \@safe@activetrue
580 \org@ifthenelse{#1}%
581 {\let\pageref\bbl@temp@pref

```

```

582         \let\ref\bbl@temp@ref
583         \@safe@activesfalse
584         #2}%
585     {\let\pageref\bbl@temp@pref
586      \let\ref\bbl@temp@ref
587      \@safe@activesfalse
588      #3}%
589     }%
590   }{}%
591 }

```

### 7.7.2 varioref

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

```

592 \AtBeginDocument{%
593   \ifpackageloaded{varioref}{%
594     \bbl@redefine\@@vpageref#1[#2]#3{%
595       \@safe@activestrue
596       \org@@@vpageref{#1}{#2}{#3}%
597       \@safe@activesfalse}%
598     \bbl@redefine\vrefpagemum#1#2{%
599       \@safe@activestrue
600       \org@vrefpagemum{#1}{#2}%
601       \@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.

```

602   \expandafter\def\csname Ref \endcsname#1{%
603     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
604   }{}%
605 }
606 \fi

```

### 7.7.3 hhline

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

```

607 \AtEndOfPackage{%
608   \AtBeginDocument{%
609     \ifpackageloaded{hhline}%
610     {\expandafter\ifx\csname normal@char\string\endcsname\relax
611       \else
612         \makeatletter
613         \def\@currname{hhline}\input{hhline.sty}\makeatother
614       \fi}%
615     {}}}

```

### 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? Commented out in 2020/07/27.

```
616% \AtBeginDocument{%
617%   \ifx\pdfstringdefDisableCommands\undefined\else
618%     \pdfstringdefDisableCommands{\languageshorthands{system}}%
619%   \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.

```
620 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
621   \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 L<sup>A</sup>T<sub>E</sub>X.

```
622 \def\substitutefontfamily#1#2#3{%
623   \lowercase{\immediate\openout15=#1#2.fd\relax}%
624   \immediate\write15{%
625     \string\ProvidesFile{#1#2.fd}%
626     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
627     \space generated font description file]^^J
628     \string\DeclareFontFamily{#1}{#2}{^^J
629     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
630     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
631     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
632     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
633     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
634     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
635     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
636     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
637   }%
638   \closeout15
639 }
640 \@onlypreamble\substitutefontfamily
```

## 7.8 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X 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`

```
641 \bbl@trace{Encoding and fonts}
642 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,PU,PD1}
643 \newcommand\BabelNonText{TS1,T3,TS3}
644 \let\org@TeX\TeX
645 \let\org@LaTeX\LaTeX
646 \let\ensureascii\@firstofone
647 \AtBeginDocument{%
648   \in@false
649   \bbl@foreach\BabelNonASCII{% is there a text non-ascii enc?
```

```

650 \ifin@ \else
651 \lowercase{\bbl@xin@{, #1 enc.def, }{, \@filelist, }}%
652 \fi}%
653 \ifin@ % if a text non-ascii has been loaded
654 \def\ensureasciic#1{{\fontencoding{OT1}\selectfont#1}}%
655 \DeclareTextCommandDefault{\TeX}{\org@TeX}%
656 \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
657 \def\bbl@tempb#1\@@{\uppercase{\bbl@tempc#1}ENC.DEF\@empty\@@}%
658 \def\bbl@tempc#1ENC.DEF#2\@@{\%
659 \ifx\@empty#2\else
660 \bbl@ifunset{T#1}%
661 {}%
662 {\bbl@xin@{, #1, }{, \BabelNonASCII, \BabelNonText, }}%
663 \ifin@
664 \DeclareTextCommand{\TeX}{#1}{\ensureasciic{\org@TeX}}%
665 \DeclareTextCommand{\LaTeX}{#1}{\ensureasciic{\org@LaTeX}}%
666 \else
667 \def\ensureasciic##1{{\fontencoding{#1}\selectfont##1}}%
668 \fi}%
669 \fi}%
670 \bbl@foreach\@filelist{\bbl@tempb#1\@@}% TODO - \@@ de mas??
671 \bbl@xin@{, \cf@encoding, }{, \BabelNonASCII, \BabelNonText,}%
672 \ifin@ \else
673 \edef\ensureasciic#1{\%
674 \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
675 \fi
676 \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.

```

677 \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.

```

678 \AtBeginDocument{%
679 \@ifpackageloaded{fontspec}%
680 {\xdef\latinencoding{%
681 \ifx\UTFencname\@undefined
682 EU\ifcase\bbl@engine\or2\or1\fi
683 \else
684 \UTFencname
685 \fi}}%
686 {\gdef\latinencoding{OT1}%
687 \ifx\cf@encoding\bbl@t@one
688 \xdef\latinencoding{\bbl@t@one}%
689 \else
690 \ifx\@fontenc@load@list\@undefined
691 \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}}%
692 \else
693 \def\@elt#1{, #1,}%
694 \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
695 \let\@elt\relax
696 \bbl@xin@{, T1, }\bbl@tempa

```



```

697         \ifin@
698         \xdef\latinencoding{\bbl@t@one}%
699         \fi
700     \fi
701 \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.

```

702 \DeclareRobustCommand{\latintext}{%
703   \fontencoding{\latinencoding}\selectfont
704   \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.

```

705 \ifx\@undefined\DeclareTextFontCommand
706   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
707 \else
708   \DeclareTextFontCommand{\textlatin}{\latintext}
709 \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 `Lua $\TeX$ -ja` 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.

```

710 \ifodd\bbl@engine
711   \def\bbl@activate@preotf{%
712     \let\bbl@activate@preotf\relax % only once
713     \directlua{
714       Babel = Babel or {}
715       %
716       function Babel.pre_otfload_v(head)
717         if Babel.numbers and Babel.digits_mapped then
718           head = Babel.numbers(head)
719         end
720         if Babel.bidi_enabled then
721           head = Babel.bidi(head, false, dir)

```

```

722     end
723     return head
724 end
725 %
726 function Babel.pre_otfload_h(head, gc, sz, pt, dir)
727     if Babel.numbers and Babel.digits_mapped then
728         head = Babel.numbers(head)
729     end
730     if Babel.bidi_enabled then
731         head = Babel.bidi(head, false, dir)
732     end
733     return head
734 end
735 %
736 luatexbase.add_to_callback('pre_linebreak_filter',
737     Babel.pre_otfload_v,
738     'Babel.pre_otfload_v',
739     luatexbase.priority_in_callback('pre_linebreak_filter',
740         'luaotfload.node_processor') or nil)
741 %
742 luatexbase.add_to_callback('hpack_filter',
743     Babel.pre_otfload_h,
744     'Babel.pre_otfload_h',
745     luatexbase.priority_in_callback('hpack_filter',
746         'luaotfload.node_processor') or nil)
747 }}
748 \fi

```

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

```

749 \bbl@trace{Loading basic (internal) bidi support}
750 \ifodd\bbl@engine
751   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
752     \let\bbl@beforeforeign\leavevmode
753     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
754     \RequirePackage{luatexbase}
755     \bbl@activate@preotf
756     \directlua{
757       require('babel-data-bidi.lua')
758       \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
759         require('babel-bidi-basic.lua')
760       \or
761         require('babel-bidi-basic-r.lua')
762     \fi}
763     % TODO - to locale_props, not as separate attribute
764     \newattribute\bbl@attr@dir
765     % TODO. I don't like it, hackish:
766     \bbl@exp{\output{\bodydir\pagedir\the\output}}
767     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
768   \fi\fi
769 \else
770   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
771     \bbl@error
772     {The bidi method 'basic' is available only in\\%
773       luatex. I'll continue with 'bidi=default', so\\%
774       expect wrong results}%
775     {See the manual for further details.}%
776     \let\bbl@beforeforeign\leavevmode
777     \AtEndOfPackage{%

```

```

778     \EnableBabelHook{babel-bidi}%
779     \bbl@xebidipar}
780 \fi\fi
781 \def\bbl@loadxebidi#1{%
782     \ifx\RTLfootnotetext\@undefined
783     \AtEndOfPackage{%
784         \EnableBabelHook{babel-bidi}%
785         \ifx\fontspec\@undefined
786             \bbl@loadfontspec % bidi needs fontspec
787         \fi
788         \usepackage#1{bidi}}%
789     \fi}
790 \ifnum\bbl@bidimode>200
791     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
792         \bbl@tentative{bidi=bidi}
793         \bbl@loadxebidi{}
794     \or
795         \bbl@loadxebidi{[rldocument]}
796     \or
797         \bbl@loadxebidi{}
798     \fi
799 \fi
800 \fi
801 \ifnum\bbl@bidimode=\@ne
802     \let\bbl@beforeforeign\leavevmode
803     \ifodd\bbl@engine
804         \newattribute\bbl@attr@dir
805         \bbl@exp{\output{\bodydir\pagedir\the\output}}%
806     \fi
807 \AtEndOfPackage{%
808     \EnableBabelHook{babel-bidi}%
809     \ifodd\bbl@engine\else
810         \bbl@xebidipar
811     \fi}
812 \fi

```

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

```

813 \bbl@trace{Macros to switch the text direction}
814 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
815 \def\bbl@rscripts{% TODO. Base on codes ??
816     ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
817     Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
818     Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
819     Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
820     Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
821     Old South Arabian,%
822 \def\bbl@provide@dirs#1{%
823     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
824     \ifin@
825         \global\bbl@csarg\chardef{wdir@#1}\@ne
826         \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
827     \ifin@
828         \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
829     \fi
830 \else
831     \global\bbl@csarg\chardef{wdir@#1}\z@
832 \fi
833 \ifodd\bbl@engine

```

```

834 \bbl@csarg\ifcase{wdir@#1}%
835 \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
836 \or
837 \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
838 \or
839 \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
840 \fi
841 \fi}
842 \def\bbl@switchdir{%
843 \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
844 \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
845 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}%
846 \def\bbl@setdirs#1{% TODO - math
847 \ifcase\bbl@select@type % TODO - strictly, not the right test
848 \bbl@bodydir{#1}%
849 \bbl@pdir{#1}%
850 \fi
851 \bbl@textdir{#1}}
852 % TODO. Only if \bbl@bidimode > 0?:
853 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
854 \DisableBabelHook{babel-bidi}

```

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

```

855 \ifodd\bbl@engine % luatex=1
856 \chardef\bbl@thetextdir\z@
857 \chardef\bbl@thepardir\z@
858 \def\bbl@getluadir#1{%
859 \directlua{
860 if tex.#1dir == 'TLT' then
861 tex.sprint('0')
862 elseif tex.#1dir == 'TRT' then
863 tex.sprint('1')
864 end}}
865 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
866 \ifcase#3\relax
867 \ifcase\bbl@getluadir{#1}\relax\else
868 #2 TLT\relax
869 \fi
870 \else
871 \ifcase\bbl@getluadir{#1}\relax
872 #2 TRT\relax
873 \fi
874 \fi}
875 \def\bbl@textdir#1{%
876 \bbl@setluadir{text}\textdir{#1}%
877 \chardef\bbl@thetextdir#1\relax
878 \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
879 \def\bbl@pdir#1{%
880 \bbl@setluadir{par}\pardir{#1}%
881 \chardef\bbl@thepardir#1\relax}
882 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
883 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
884 \def\bbl@dirparastext{\pardir\the\textdir\relax}% %%%
885 % Sadly, we have to deal with boxes in math with basic.
886 % Activated every math with the package option bidi=:
887 \def\bbl@mathboxdir{%
888 \ifcase\bbl@thetextdir\relax
889 \everyhbox{\textdir TLT\relax}%
890 \else

```

```

891 \everyhbox{\textdir TRT\relax}%
892 \fi}
893 \frozen@everymath\expandafter{%
894 \expandafter\bb1@mathboxdir\the\frozen@everymath}
895 \frozen@everydisplay\expandafter{%
896 \expandafter\bb1@mathboxdir\the\frozen@everydisplay}
897 \else % pdftex=0, xetex=2
898 \newcount\bb1@dirlevel
899 \chardef\bb1@thetextdir\z@
900 \chardef\bb1@thepardir\z@
901 \def\bb1@textdir#1{%
902 \ifcase#1\relax
903 \chardef\bb1@thetextdir\z@
904 \bb1@textdir@i\beginL\endL
905 \else
906 \chardef\bb1@thetextdir@ne
907 \bb1@textdir@i\beginR\endR
908 \fi}
909 \def\bb1@textdir@i#1#2{%
910 \ifhmode
911 \ifnum\currentgrouplevel>\z@
912 \ifnum\currentgrouplevel=\bb1@dirlevel
913 \bb1@error{Multiple bidi settings inside a group}%
914 {I'll insert a new group, but expect wrong results.}%
915 \bgroup\aftergroup#2\aftergroup\egroup
916 \else
917 \ifcase\currentgrouptype\or % 0 bottom
918 \aftergroup#2% 1 simple {}
919 \or
920 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
921 \or
922 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
923 \or\or\or % vbox vtop align
924 \or
925 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
926 \or\or\or\or\or\or % output math disc insert vcent mathchoice
927 \or
928 \aftergroup#2% 14 \begingroup
929 \else
930 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
931 \fi
932 \fi
933 \bb1@dirlevel\currentgrouplevel
934 \fi
935 #1%
936 \fi}
937 \def\bb1@pardir#1{\chardef\bb1@thepardir#1\relax}
938 \let\bb1@bodydir@gobble
939 \let\bb1@pagedir@gobble
940 \def\bb1@dirparastext{\chardef\bb1@thepardir\bb1@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).

```

941 \def\bb1@xebidipar{%
942 \let\bb1@xebidipar\relax
943 \TeXeTstate\@ne
944 \def\bb1@xeverypar{%
945 \ifcase\bb1@thepardir

```

```

946     \ifcase\bbl@thetextdir\else\beginR\fi
947     \else
948     {\setbox\z@\lastbox\beginR\box\z@}%
949     \fi}%
950     \let\bbl@severypar\everypar
951     \newtoks\everypar
952     \everypar=\bbl@severypar
953     \bbl@severypar{\bbl@xeverypar\the\everypar}}
954 \ifnum\bbl@bidimode>200
955     \let\bbl@textdir=i@gobbletwo
956     \let\bbl@xebidipar\@empty
957     \AddBabelHook{bidi}{foreign}{%
958         \def\bbl@tempa{\def\BabelText####1}%
959         \ifcase\bbl@thetextdir
960             \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
961         \else
962             \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
963         \fi}
964     \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
965 \fi
966 \fi

```

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

```

967 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
968 \AtBeginDocument{%
969     \ifx\pdfstringdefDisableCommands\@undefined\else
970         \ifx\pdfstringdefDisableCommands\relax\else
971             \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
972         \fi
973     \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`.

```

974 \bbl@trace{Local Language Configuration}
975 \ifx\loadlocalcfg\@undefined
976     \@ifpackagewith{babel}{noconfigs}%
977     {\let\loadlocalcfg@gobble}%
978     {\def\loadlocalcfg#1{%
979         \InputIfFileExists{#1.cfg}%
980         {\typeout{*****^J%
981             * Local config file #1.cfg used^^J%
982             *}}%
983         \@empty}}
984 \fi

```

Just to be compatible with  $\text{\TeX}$  2.09 we add a few more lines of code. TODO. Necessary? Correct place? Used by some ldf file?

```

985 \ifx\@unexpandable@protect\@undefined
986     \def\@unexpandable@protect{\noexpand\protect\noexpand}
987     \long\def\protected@write#1#2#3{%
988         \begingroup
989         \let\thepage\relax
990         #2%

```

```

991      \let\protect\@unexpandable@protect
992      \edef\reserved@a{\write#1{#3}}%
993      \reserved@a
994      \endgroup
995      \if@nobreak\ifvmode\nobreak\fi\fi}
996 \fi
997 %
998 % \subsection{Language options}
999 %
1000 % Languages are loaded when processing the corresponding option
1001 % \textit{except} if a |main| language has been set. In such a
1002 % case, it is not loaded until all options has been processed.
1003 % The following macro inputs the ldf file and does some additional
1004 % checks (|\input| works, too, but possible errors are not caught).
1005 %
1006 %      \begin{macrocode}
1007 \bbl@trace{Language options}
1008 \let\bbl@afterlang\relax
1009 \let\BabelModifiers\relax
1010 \let\bbl@loaded\@empty
1011 \def\bbl@load@language#1{%
1012   \InputIfFileExists{#1.ldf}%
1013   {\edef\bbl@loaded{\CurrentOption
1014     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
1015     \expandafter\let\expandafter\bbl@afterlang
1016       \csname\CurrentOption.ldf-h@@k\endcsname
1017     \expandafter\let\expandafter\BabelModifiers
1018       \csname bbl@mod@\CurrentOption\endcsname}%
1019   {\bbl@error{%
1020     Unknown option '\CurrentOption'. Either you misspelled it\\
1021     or the language definition file \CurrentOption.ldf was not found}}%
1022     Valid options are, among others: shorthands=, KeepShorthandsActive,\\
1023     activeacute, activegrave, noconfigs, safe=, main=, math=\\
1024     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.

```

1025 \def\bbl@try@load@lang#1#2#3{%
1026   \IfFileExists{\CurrentOption.ldf}%
1027   {\bbl@load@language{\CurrentOption}}%
1028   {#1\bbl@load@language{#2}#3}}
1029 \DeclareOption{hebrew}{%
1030   \input{rlbabel.def}%
1031   \bbl@load@language{hebrew}}
1032 \DeclareOption{hungarian}{\bbl@try@load@lang{\magyar}{}}
1033 \DeclareOption{lowersorbian}{\bbl@try@load@lang{\lsorbian}{}}
1034 \DeclareOption{nynorsk}{\bbl@try@load@lang{\norsk}{}}
1035 \DeclareOption{polutonikogreek}{%
1036   \bbl@try@load@lang{\greek}{\languageattribute{greek}{polutoniko}}}
1037 \DeclareOption{russian}{\bbl@try@load@lang{\russianb}{}}
1038 \DeclareOption{ukrainian}{\bbl@try@load@lang{\ukraineb}{}}
1039 \DeclareOption{uppersorbian}{\bbl@try@load@lang{\usorbian}{}}

```

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

```

1040 \ifx\bbl@opt@config\@nnil

```

```

1041 \ifpackagewith{babel}{noconfigs}{}%
1042 {\InputIfFileExists{bblopts.cfg}%
1043 {\typeout{*****^J%
1044          * Local config file bblopts.cfg used^^J%
1045          *}}%
1046 }%
1047 \else
1048 \InputIfFileExists{\bbl@opt@config.cfg}%
1049 {\typeout{*****^J%
1050          * Local config file \bbl@opt@config.cfg used^^J%
1051          *}}%
1052 {\bbl@error{%
1053     Local config file '\bbl@opt@config.cfg' not found}%
1054     Perhaps you misspelled it.}%
1055 \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.

```

1056 \let\bbl@tempc\relax
1057 \bbl@foreach\bbl@language@opts{%
1058   \ifcase\bbl@iniflag % Default
1059     \bbl@ifunset{ds@#1}%
1060     {\DeclareOption{#1}{\bbl@load@language{#1}}}%
1061     {}%
1062   \or % provide=*
1063     \@gobble % case 2 same as 1
1064   \or % provide+=*
1065     \bbl@ifunset{ds@#1}%
1066     {\IfFileExists{#1.ldf}{}%
1067      {\IfFileExists{babel-#1.tex}{\@namedef{ds@#1}}}%
1068      {}%
1069     \bbl@ifunset{ds@#1}%
1070     {\def\bbl@tempc{#1}%
1071      \DeclareOption{#1}{%
1072        \ifnum\bbl@iniflag>\@ne
1073          \bbl@ldfinit
1074          \babelprovide[import]{#1}%
1075          \bbl@afterldf}%
1076        \else
1077          \bbl@load@language{#1}%
1078        \fi}%
1079      }%
1080   \or % provide*=*
1081     \def\bbl@tempc{#1}%
1082     \bbl@ifunset{ds@#1}%
1083     {\DeclareOption{#1}{%
1084       \bbl@ldfinit
1085       \babelprovide[import]{#1}%
1086       \bbl@afterldf}%
1087     }%
1088   \fi}

```

Now, we make sure an option is explicitly declared for any language set as global option, by checking if an `ldf` exists. The previous step was, in fact, somewhat redundant, but that way we minimize accessing the file system just to see if the option could be a language.

```

1089 \let\bbl@tempb\@nnil
1090 \bbl@foreach\@classoptionslist{%

```



```

1091 \bbl@ifunset{ds@#1}%
1092 {\IfFileExists{#1.ldf}}}%
1093 {\IfFileExists{babel-#1.tex}}{\@namedef{ds@#1}}}%
1094 {}%
1095 \bbl@ifunset{ds@#1}%
1096 {\def\bbl@tempb{#1}%
1097 \DeclareOption{#1}{%
1098 \ifnum\bbl@iniflag>\@ne
1099 \bbl@ldfinit
1100 \babelprovide[import]{#1}%
1101 \bbl@afterldf}%
1102 \else
1103 \bbl@load@language{#1}%
1104 \fi}}%
1105 {}%

```

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

```

1106 \ifnum\bbl@iniflag=\z@\else
1107 \ifx\bbl@opt@main\@nnil
1108 \ifx\bbl@tempc\relax
1109 \let\bbl@opt@main\bbl@tempb
1110 \else
1111 \let\bbl@opt@main\bbl@tempc
1112 \fi
1113 \fi
1114 \fi
1115 \ifx\bbl@opt@main\@nnil\else
1116 \expandafter
1117 \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
1118 \expandafter\let\csname ds@\bbl@opt@main\endcsname\@empty
1119 \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  $\TeX$  processes before):

```

1120 \def\AfterBabelLanguage#1{%
1121 \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}}%
1122 \DeclareOption*{}
1123 \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.

```

1124 \bbl@trace{Option 'main'}
1125 \ifx\bbl@opt@main\@nnil
1126 \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
1127 \let\bbl@tempc\@empty
1128 \bbl@for\bbl@tempb\bbl@tempa{%
1129 \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
1130 \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
1131 \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
1132 \expandafter\bbl@tempa\bbl@loaded,\@nnil
1133 \ifx\bbl@tempb\bbl@tempc\else
1134 \bbl@warning{%
1135 Last declared language option is '\bbl@tempc',\%
1136 but the last processed one was '\bbl@tempb'.\%
1137 The main language cannot be set as both a global\%

```

```

1138      and a package option. Use `main=\bbl@tempc' as\%
1139      option. Reported}%
1140 \fi
1141 \else
1142 \ifodd\bbl@iniflag % case 1,3
1143 \bbl@ldfinit
1144 \let\CurrentOption\bbl@opt@main
1145 \bbl@exp{\bbl@babelprovide[import,main]{\bbl@opt@main}}
1146 \bbl@afterldf}%
1147 \else % case 0,2
1148 \chardef\bbl@iniflag\z@ % Force ldf
1149 \expandafter\let\csname ds@\bbl@opt@main\endcsname\bbl@loadmain
1150 \ExecuteOptions{\bbl@opt@main}
1151 \DeclareOption*{}%
1152 \ProcessOptions*
1153 \fi
1154 \fi
1155 \def\AfterBabelLanguage{%
1156 \bbl@error
1157 {Too late for \string\AfterBabelLanguage}%
1158 {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.

1159 \ifx\bbl@main@language\undefined
1160 \bbl@info{%
1161 You haven't specified a language. I'll use 'nil'\%
1162 as the main language. Reported}
1163 \bbl@load@language{nil}
1164 \fi
1165 \</package>
1166 \*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

```

1167 \ifx\ldf@quit\undefined\else
1168 \endinput\fi % Same line!
1169 <<Make sure ProvidesFile is defined>>
1170 \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 an alternative mechanism is provided. For the moment, only \babeloptionstrings and \babeloptionmath are provided, which can be defined before loading babel. \BabelModifiers can be set too (but not sure it works).

```

1171 \ifx\AtBeginDocument\@undefined % TODO. change test.
1172 <<Emulate LaTeX>>
1173 \def\language{english}%
1174 \let\bbl@opt@shorthands\@nnil
1175 \def\bbl@ifshorthand#1#2#3{#2}%
1176 \let\bbl@language@opts\@empty
1177 \ifx\babeloptionstrings\@undefined
1178   \let\bbl@opt@strings\@nnil
1179 \else
1180   \let\bbl@opt@strings\babeloptionstrings
1181 \fi
1182 \def\BabelStringsDefault{generic}
1183 \def\bbl@tempa{normal}
1184 \ifx\babeloptionmath\bbl@tempa
1185   \def\bbl@mathnormal{\noexpand\textormath}
1186 \fi
1187 \def\AfterBabelLanguage#1#2{}
1188 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
1189 \let\bbl@afterlang\relax
1190 \def\bbl@opt@safe{BR}
1191 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
1192 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
1193 \expandafter\newif\csname ifbbl@single\endcsname
1194 \chardef\bbl@bidimode\z@
1195 \fi

```

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

```

1196 \ifx\bbl@trace\@undefined
1197   \let\LdfInit\endinput
1198 \def\ProvidesLanguage#1{\endinput}
1199 \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.

```

1200 <<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.

```

1201 \def\bbl@version{<<version>>}
1202 \def\bbl@date{<<date>>}
1203 \def\adddialect#1#2{%
1204   \global\chardef#1#2\relax
1205   \bbl@usehooks{adddialect}{#1}{#2}%
1206   \begingroup
1207     \count#1\relax
1208     \def\bbl@elt###1##2###3###4{%
1209       \ifnum\count@=##2\relax
1210         \bbl@info{string#1 = using hyphenrules for ##1\\%
1211           (\string\language\the\count@)}%
1212         \def\bbl@elt####1#####2####3####4{%
1213           \fi}%
1214       \bbl@cs{languages}%
1215     \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.

```

1216 \def\bbl@fixname#1{%
1217   \begingroup
1218     \def\bbl@tempe{l@}%
1219     \edef\bbl@tempd{\noexpand\ifundefined{\noexpand\bbl@tempe#1}}%
1220     \bbl@tempd
1221       {\lowercase\expandafter{\bbl@tempd}%
1222        {\uppercase\expandafter{\bbl@tempd}%
1223         \@empty
1224          {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1225           \uppercase\expandafter{\bbl@tempd}}}%
1226          {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1227           \lowercase\expandafter{\bbl@tempd}}}%
1228         \@empty
1229       \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
1230   \bbl@tempd
1231   \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}
1232 \def\bbl@iflanguage#1{%
1233   \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.

1234 \def\bbl@bcpcase#1#2#3#4\@@#5{%
1235   \ifx\@empty#3%
1236     \uppercase{\def#5{#1#2}}%
1237   \else
1238     \uppercase{\def#5{#1}}%
1239     \lowercase{\def#5{#5#2#3#4}}%
1240   \fi}
1241 \def\bbl@bcpllookup#1-#2-#3-#4\@@{%
1242   \let\bbl@bcp\relax
1243   \lowercase{\def\bbl@tempa{#1}}%
1244   \ifx\@empty#2%
1245     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1246   \else\ifx\@empty#3%
1247     \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
1248     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
1249       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
1250     {}%
1251   \ifx\bbl@bcp\relax
1252     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1253   \fi
1254   \else
1255     \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
1256     \bbl@bcpcase#3\@empty\@empty\@@\bbl@tempc
1257     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
1258       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
1259     {}%
1260   \ifx\bbl@bcp\relax
1261     \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1262     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%

```

```

1263     {}%
1264 \fi
1265 \ifx\bbbl@bcp\relax
1266   \IfFileExists{babel-\bbbl@tempa-\bbbl@tempc.ini}%
1267   {\edef\bbbl@bcp{\bbbl@tempa-\bbbl@tempc}}%
1268   {}%
1269 \fi
1270 \ifx\bbbl@bcp\relax
1271   \IfFileExists{babel-\bbbl@tempa.ini}{\let\bbbl@bcp\bbbl@tempa}{}%
1272 \fi
1273 \fi\fi}
1274 \let\bbbl@initoload\relax
1275 \def\bbbl@provide@locale{%
1276   \ifx\babelprovide\undefined
1277     \bbbl@error{For a language to be defined on the fly 'base'\\%
1278               is not enough, and the whole package must be\\%
1279               loaded. Either delete the 'base' option or\\%
1280               request the languages explicitly}%
1281     {See the manual for further details.}%
1282 \fi
1283 % TODO. Option to search if loaded, with \LocaleForEach
1284 \let\bbbl@auxname\language % Still necessary. TODO
1285 \bbbl@ifunset{\bbbl@bcp@map@\language}{}% Move uplevel??
1286 {\edef\language{\@nameuse{\bbbl@bcp@map@\language}}}%
1287 \ifbbbl@bcpallowed
1288   \expandafter\ifx\csname date\language\endcsname\relax
1289     \expandafter
1290     \bbbl@bcplookup\language-\@empty-\@empty-\@empty\@@
1291     \ifx\bbbl@bcp\relax\else % Returned by \bbbl@bcplookup
1292       \edef\language{\bbbl@bcp@prefix\bbbl@bcp}%
1293       \edef\localename{\bbbl@bcp@prefix\bbbl@bcp}%
1294       \expandafter\ifx\csname date\language\endcsname\relax
1295         \let\bbbl@initoload\bbbl@bcp
1296         \bbbl@exp{\babelprovide[\bbbl@autoload@bcptoptions]{\language}}%
1297         \let\bbbl@initoload\relax
1298       \fi
1299       \bbbl@csarg\xdef{bcp@map@\bbbl@bcp}{\localename}%
1300     \fi
1301   \fi
1302 \fi
1303 \expandafter\ifx\csname date\language\endcsname\relax
1304   \IfFileExists{babel-\language.tex}%
1305   {\bbbl@exp{\babelprovide[\bbbl@autoload@options]{\language}}}%
1306   {}%
1307 \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.

```

1308 \def\iflanguage#1{%
1309   \bbbl@iflanguage{#1}%
1310   \ifnum\csname l@#1\endcsname=\language
1311     \expandafter\@firstoftwo
1312   \else
1313     \expandafter\@secondoftwo
1314   \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.

```
1315 \let\bbl@select@type\z@
1316 \edef\selectlanguage{%
1317   \noexpand\protect
1318   \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`.

```
1319 \ifx\@undefined\protect\let\protect\relax\fi
```

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

```
1320 \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.

```
1321 \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`

```
1322 \def\bbl@push@language{%
1323   \ifx\language\@undefined\else
1324     \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1325   \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 in `\bbl@language@stack`.

```
1326 \def\bbl@pop@lang#1+#2\@@{%
1327   \edef\language{#1}%
1328   \xdef\bbl@language@stack{#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).

```
1329 \let\bbl@ifrestoring\@secondoftwo
1330 \def\bbl@pop@language{%
1331   \expandafter\bbl@pop@lang\bbl@language@stack\@@
1332   \let\bbl@ifrestoring\@firstoftwo
1333   \expandafter\bbl@set@language\expandafter{\language}%
1334   \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).

```

1335 \chardef\localeid\z@
1336 \def\bbl@id@last{0}    % No real need for a new counter
1337 \def\bbl@id@assign{%
1338   \bbl@ifunset{bbl@id@@\language}%
1339   {\count@bbl@id@last\relax
1340     \advance\count@ne
1341     \bbl@csarg\chardef{id@@\language}\count@
1342     \edef\bbl@id@last{\the\count@}%
1343     \ifcase\bbl@engine\or
1344       \directlua{
1345         Babel = Babel or {}
1346         Babel.locale_props = Babel.locale_props or {}
1347         Babel.locale_props[\bbl@id@last] = {}
1348         Babel.locale_props[\bbl@id@last].name = '\language'
1349       }%
1350     \fi}%
1351   }%
1352   \chardef\localeid\bbl@c{l{id@}}

```

The unprotected part of `\selectlanguage`.

```

1353 \expandafter\def\csname selectlanguage \endcsname#1{%
1354   \ifnum\bbl@hymapsel=\@ccclv\let\bbl@hymapsel\tw@fi
1355   \bbl@push@language
1356   \aftergroup\bbl@pop@language
1357   \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.

```

1358 \def\BabelContentsFiles{toc,lof,lot}
1359 \def\bbl@set@language#1{% from selectlanguage, pop@
1360   % The old buggy way. Preserved for compatibility.
1361   \edef\language{%
1362     \ifnum\escapechar=\expandafter`\string#1\@empty
1363     \else\string#1\@empty\fi}%
1364   \ifcat\relax\noexpand#1%
1365     \expandafter\ifx\csname date\language\endcsname\relax
1366       \edef\language{#1}%
1367       \let\localename\language
1368     \else
1369       \bbl@info{Using '\string\language' instead of 'language' is\\%
1370         deprecated. If what you want is to use a\\%
1371         macro containing the actual locale, make\\%
1372         sure it does not match any language.\\%
1373         Reported}%
1374       I'll\\%
1375       try to fix '\string\localename', but I cannot promise\\%

```

```

1376%                anything. Reported}%
1377    \ifx\scantokens\undefined
1378        \def\localename{??}%
1379    \else
1380        \scantokens\expandafter{\expandafter
1381            \def\expandafter\localename\expandafter{\language}%
1382        \fi
1383    \fi
1384 \else
1385     \def\localename{#1}% This one has the correct catcodes
1386 \fi
1387 \select@language{\language}%
1388 % write to auxs
1389 \expandafter\ifx\csname date\language\endcsname\relax\else
1390     \if@filesw
1391         \ifx\babel@aux\gobbletwo\else % Set if single in the first, redundant
1392             % \bbl@savelastskip
1393             \protected@write\auxout{}\string\babel@aux{\bbl@auxname}{}}%
1394             % \bbl@restorelastskip
1395         \fi
1396         \bbl@usehooks{write}}}%
1397     \fi
1398 \fi}
1399 % The following is used above to deal with skips before the write
1400 % whatsit. Adapted from hyperref, but it might fail, so for the moment
1401 % it's not activated. TODO.
1402 \def\bbl@savelastskip{%
1403     \let\bbl@restorelastskip\relax
1404     \ifvmode
1405         \ifdim\lastskip=\z@
1406             \let\bbl@restorelastskip\nobreak
1407         \else
1408             \bbl@exp{%
1409                 \def\\bbl@restorelastskip{%
1410                     \skip@=\the\lastskip
1411                     \\nobreak \vskip-\skip@ \vskip\skip@}}%
1412             \fi
1413         \fi}
1414 \newif\ifbbl@bcpallowed
1415 \bbl@bcpallowedfalse
1416 \def\select@language#1{% from set@, babel@aux
1417     % set hymap
1418     \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1419     % set name
1420     \edef\language{#1}%
1421     \bbl@fixname\language
1422     % TODO. name@map must be here?
1423     \bbl@provide@locale
1424     \bbl@iflanguage\language{%
1425         \expandafter\ifx\csname date\language\endcsname\relax
1426             \bbl@error
1427             {Unknown language '\language'. Either you have\\%
1428              misspelled its name, it has not been installed,\\%
1429              or you requested it in a previous run. Fix its name,\\%
1430              install it or just rerun the file, respectively. In\\%
1431              some cases, you may need to remove the aux file}%
1432             {You may proceed, but expect wrong results}%
1433         \else
1434             % set type

```



```

1435 \let\bbl@select@type\z@
1436 \expandafter\bbl@switch\expandafter{\language}%
1437 \fi}}
1438 \def\babel@aux#1#2{% TODO. See how to avoid undefined nil's
1439 \select@language{#1}%
1440 \bbl@foreach\BabelContentsFiles{%
1441 \@writefile{##1}{\babel@toc{#1}{#2}}}% % TODO - ok in plain?
1442 \def\babel@toc#1#2{%
1443 \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.

```

1444 \newif\ifbbl@usedategroup
1445 \def\bbl@switch#1{% from select@, foreign@
1446 % make sure there is info for the language if so requested
1447 \bbl@ensureinfo{#1}%
1448 % restore
1449 \originalTeX
1450 \expandafter\def\expandafter\originalTeX\expandafter{%
1451 \csname noextras#1\endcsname
1452 \let\originalTeX\@empty
1453 \babel@beginsave}%
1454 \bbl@usehooks{afterreset}{}%
1455 \languageshorthands{none}%
1456 % set the locale id
1457 \bbl@id@assign
1458 % switch captions, date
1459 % No text is supposed to be added here, so we remove any
1460 % spurious spaces.
1461 \bbl@bsphack
1462 \ifcase\bbl@select@type
1463 \csname captions#1\endcsname\relax
1464 \csname date#1\endcsname\relax
1465 \else
1466 \bbl@xin@{,captions,}{, \bbl@select@opts,}%
1467 \ifin@
1468 \csname captions#1\endcsname\relax
1469 \fi
1470 \bbl@xin@{,date,}{, \bbl@select@opts,}%
1471 \ifin@ % if \foreign... within \<lang>date
1472 \csname date#1\endcsname\relax
1473 \fi
1474 \fi
1475 \bbl@esphack
1476 % switch extras
1477 \bbl@usehooks{beforeextras}{}%
1478 \csname extras#1\endcsname\relax
1479 \bbl@usehooks{afterextras}{}%
1480 % > babel-ensure

```

```

1481 % > babel-sh-<short>
1482 % > babel-bidi
1483 % > babel-fontspec
1484 % hyphenation - case mapping
1485 \ifcase\bbbl@opt@hyphenmap\or
1486   \def\BabelLower##1##2{\lcode##1=##2\relax}%
1487   \ifnum\bbbl@hymapsel>4\else
1488     \csname\language @\bbbl@hyphenmap\endcsname
1489   \fi
1490   \chardef\bbbl@opt@hyphenmap\z@
1491 \else
1492   \ifnum\bbbl@hymapsel>\bbbl@opt@hyphenmap\else
1493     \csname\language @\bbbl@hyphenmap\endcsname
1494   \fi
1495 \fi
1496 \let\bbbl@hymapsel\@cclv
1497 % hyphenation - select patterns
1498 \bbbl@patterns{#1}%
1499 % hyphenation - allow stretching with babelnohyphens
1500 \ifnum\language=\l@babelnohyphens
1501   \babel@savevariable\emergencystretch
1502   \emergencystretch\maxdimen
1503   \babel@savevariable\hbadness
1504   \hbadness\@M
1505 \fi
1506 % hyphenation - mins
1507 \babel@savevariable\lefthyphenmin
1508 \babel@savevariable\righthyphenmin
1509 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1510   \set@hyphenmins\tw@\thr@@\relax
1511 \else
1512   \expandafter\expandafter\expandafter\set@hyphenmins
1513     \csname #1hyphenmins\endcsname\relax
1514 \fi}

```

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

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

```

1515 \long\def\otherlanguage#1{%
1516   \ifnum\bbbl@hymapsel=\@cclv\let\bbbl@hymapsel\thr@@\fi
1517   \csname selectlanguage \endcsname{#1}%
1518   \ignorespaces}

```

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

```

1519 \long\def\endotherlanguage{%
1520   \global\@ignoretrue\ignorespaces}

```

`otherlanguage*` The `otherlanguage` environment is meant to be used when a large part of text from a different language needs to be typeset, but without changing the translation of words such as ‘figure’. This environment makes use of `\foreign@language`.

```

1521 \expandafter\def\csname otherlanguage*\endcsname{%
1522   \@ifnextchar[\bbbl@otherlanguage@s{\bbbl@otherlanguage@s[]}%
1523   \def\bbbl@otherlanguage@s[#1]#2{%
1524     \ifnum\bbbl@hymapsel=\@cclv\chardef\bbbl@hymapsel4\relax\fi
1525     \def\bbbl@select@opts{#1}%

```

```
1526 \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”.

```
1527 \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`.

```
1528 \providecommand\bbl@beforeforeign{}
1529 \edef\foreignlanguage{%
1530   \noexpand\protect
1531   \expandafter\noexpand\csname foreignlanguage \endcsname}
1532 \expandafter\def\csname foreignlanguage \endcsname{%
1533   \@ifstar\bbl@foreign@s\bbl@foreign@x}
1534 \providecommand\bbl@foreign@x[3][]{%
1535   \begingroup
1536     \def\bbl@select@opts{#1}%
1537     \let\BabelText\@firstofone
1538     \bbl@beforeforeign
1539     \foreign@language{#2}%
1540     \bbl@usehooks{foreign}{}%
1541     \BabelText{#3}% Now in horizontal mode!
1542   \endgroup}
1543 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
1544   \begingroup
1545     {\par}%
1546     \let\BabelText\@firstofone
1547     \foreign@language{#1}%
1548     \bbl@usehooks{foreign*}{}%
1549     \bbl@dirparastext
1550     \BabelText{#2}% Still in vertical mode!
1551     {\par}%
1552   \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`.

```
1553 \def\foreign@language#1{%
1554   % set name
1555   \edef\language{#1}%
```

```

1556 \ifbbl@usedategroup
1557   \bbl@add\bbl@select@opts{,date,}%
1558   \bbl@usedategroupfalse
1559 \fi
1560 \bbl@fixname\language
1561 % TODO. name@map here?
1562 \bbl@provide@locale
1563 \bbl@iflanguage\language{
1564   \expandafter\ifx\csname date\language\endcsname\relax
1565     \bbl@warning % TODO - why a warning, not an error?
1566     {Unknown language `#1'. Either you have\\%
1567       misspelled its name, it has not been installed,\\%
1568       or you requested it in a previous run. Fix its name,\\%
1569       install it or just rerun the file, respectively. In\\%
1570       some cases, you may need to remove the aux file.\\%
1571       I'll proceed, but expect wrong results.\\%
1572       Reported}%
1573   \fi
1574   % set type
1575   \let\bbl@select@type\@ne
1576   \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.

```

1577 \let\bbl@hyphlist\@empty
1578 \let\bbl@hyphenation@\relax
1579 \let\bbl@pttnlist\@empty
1580 \let\bbl@patterns@\relax
1581 \let\bbl@hymapsel=\@cclv
1582 \def\bbl@patterns#1{%
1583   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1584     \csname l@#1\endcsname
1585     \edef\bbl@tempa{#1}%
1586   \else
1587     \csname l@#1:\f@encoding\endcsname
1588     \edef\bbl@tempa{#1:\f@encoding}%
1589   \fi
1590   \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}}%
1591 % > luatex
1592 \@ifundefined{bbl@hyphenation@}{% Can be \relax!
1593   \begingroup
1594     \bbl@xin@{, \number\language,}{, \bbl@hyphlist}%
1595     \ifin@ \else
1596       \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}%
1597       \hyphenation{%
1598         \bbl@hyphenation@
1599         \@ifundefined{bbl@hyphenation@#1}%
1600         \@empty
1601         {\space\csname bbl@hyphenation@#1\endcsname}}%
1602       \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1603     \fi
1604   \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 `otherlanguage*`.

```

1605 \def\hyphenrules#1{%
1606   \edef\bbl@tempf{#1}%
1607   \bbl@fixname\bbl@tempf
1608   \bbl@iflanguage\bbl@tempf{%
1609     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1610     \ifx\languageshorthands\undefined\else
1611       \languageshorthands{none}%
1612     \fi
1613     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1614       \set@hyphenmins\tw@\thr@@\relax
1615     \else
1616       \expandafter\expandafter\expandafter\set@hyphenmins
1617       \csname\bbl@tempf hyphenmins\endcsname\relax
1618     \fi}}
1619 \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.

```

1620 \def\providehyphenmins#1#2{%
1621   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1622     \@namedef{#1hyphenmins}{#2}%
1623   \fi}

```

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

```

1624 \def\set@hyphenmins#1#2{%
1625   \lefthyphenmin#1\relax
1626   \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.

```

1627 \ifx\ProvidesFile\undefined
1628   \def\ProvidesLanguage#1[#2 #3 #4]{%
1629     \wlog{Language: #1 #4 #3 <#2>}%
1630   }
1631 \else
1632   \def\ProvidesLanguage#1{%
1633     \begingroup
1634     \catcode`\ 10 %
1635     \@makeother\/%
1636     \@ifnextchar[%]
1637       {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
1638   \def\@provideslanguage#1[#2]{%
1639     \wlog{Language: #1 #2}%
1640     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1641   \endgroup}
1642 \fi

```

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

```

1643 \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.

```
1644 \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’:

```
1645 \providecommand\setlocale{%
1646   \bbl@error
1647   {Not yet available}%
1648   {Find an armchair, sit down and wait}}
1649 \let\uselocale\setlocale
1650 \let\locale\setlocale
1651 \let\selectlocale\setlocale
1652 \let\localename\setlocale
1653 \let\textlocale\setlocale
1654 \let\textlanguage\setlocale
1655 \let\language\setlocale
```

## 9.2 Errors

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

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

```
1656 \edef\bbl@nulllanguage{\string\language=0}
1657 \ifx\PackageError\@undefined % TODO. Move to Plain
1658   \def\bbl@error#1#2{%
1659     \begingroup
1660       \newlinechar=`^^J
1661       \def\{^^J(babel) }%
1662       \errhelp{#2}\errmessage{\{#1}%
1663     \endgroup}
1664   \def\bbl@warning#1{%
1665     \begingroup
1666       \newlinechar=`^^J
1667       \def\{^^J(babel) }%
1668       \message{\{#1}%
1669     \endgroup}
1670   \let\bbl@infowarn\bbl@warning
1671   \def\bbl@info#1{%
1672     \begingroup
1673       \newlinechar=`^^J
1674       \def\{^^J}%
1675       \wlog{#1}%
1676     \endgroup}
1677 \fi
1678 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1679 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1680   \global\@namedef{#2}{\textbf{?#1?}}%
1681   \@nameuse{#2}%
1682   \bbl@warning{%
1683     \@backslashchar#2 not set. Please, define it\%
```

```

1684   after the language has been loaded (typically\\%
1685   in the preamble) with the help of\\%
1686   \string\setlocalecaption. An example is\\%
1687   \string\setlocalecaption{mylang}{chapter}{..}\\%
1688   Reported}}
1689 \def\bbl@tentative{\protect\bbl@tentative@i}
1690 \def\bbl@tentative@i#1{%
1691   \bbl@warning{%
1692     Some functions for '#1' are tentative.\\%
1693     They might not work as expected and their behavior\\%
1694     could change in the future.\\%
1695     Reported}}
1696 \def\@nolanerr#1{%
1697   \bbl@error
1698   {You haven't defined the language #1\space yet.\\%
1699   Perhaps you misspelled it or your installation\\%
1700   is not complete}%
1701   {Your command will be ignored, type <return> to proceed}}
1702 \def\@nopatterns#1{%
1703   \bbl@warning
1704   {No hyphenation patterns were preloaded for\\%
1705   the language '#1' into the format.\\%
1706   Please, configure your TeX system to add them and\\%
1707   rebuild the format. Now I will use the patterns\\%
1708   preloaded for \bbl@nulllanguage\space instead}}
1709 \let\bbl@usehooks\@gobbletwo
1710 \ifx\bbl@onlyswitch\@empty\endinput\fi
1711 % Here ended switch.def

    Here ended switch.def.

1712 \ifx\directlua\@undefined\else
1713   \ifx\bbl@luapatterns\@undefined
1714     \input luababel.def
1715   \fi
1716 \fi
1717 <<Basic macros>>
1718 \bbl@trace{Compatibility with language.def}
1719 \ifx\bbl@languages\@undefined
1720   \ifx\directlua\@undefined
1721     \openin1 = language.def % TODO. Remove hardcoded number
1722     \ifeof1
1723       \closein1
1724       \message{I couldn't find the file language.def}
1725     \else
1726       \closein1
1727       \begingroup
1728         \def\addlanguage#1#2#3#4#5{%
1729           \expandafter\ifx\csname lang@#1\endcsname\relax\else
1730             \global\expandafter\let\csname l@#1\endcsname
1731             \csname lang@#1\endcsname
1732           \fi}%
1733         \def\uselanguage#1{%
1734           \input language.def
1735         \endgroup
1736       \fi
1737     \fi
1738     \chardef\l@english\z@
1739 \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.

```

1740 \def\addto#1#2{%
1741   \ifx#1\undefined
1742     \def#1{#2}%
1743   \else
1744     \ifx#1\relax
1745       \def#1{#2}%
1746     \else
1747       {\toks@\expandafter{#1#2}%
1748        \xdef#1{\the\toks@}}%
1749   \fi
1750 \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?

```

1751 \def\bbl@withactive#1#2{%
1752   \begingroup
1753   \lccode`~=`#2\relax
1754   \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`.

```

1755 \def\bbl@redefine#1{%
1756   \edef\bbl@tempa{\bbl@stripslash#1}%
1757   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1758   \expandafter\def\csname\bbl@tempa\endcsname{
1759   \@onlypreamble\bbl@redefine

```

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

```

1760 \def\bbl@redefine@long#1{%
1761   \edef\bbl@tempa{\bbl@stripslash#1}%
1762   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1763   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname{
1764   \@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_`.

```

1765 \def\bbl@redefineroobust#1{%
1766   \edef\bbl@tempa{\bbl@stripslash#1}%
1767   \bbl@ifunset{\bbl@tempa\space}%
1768   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1769    \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1770   {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1771   \@namedef{\bbl@tempa\space}}
1772 \@onlypreamble\bbl@redefineroobust

```

### 9.3 Hooks

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



```

1773 \bbl@trace{Hooks}
1774 \newcommand\AddBabelHook[3][]{%
1775   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1776   \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1777   \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1778   \bbl@ifunset{bbl@ev@#2@#3@#1}%
1779     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1780     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1781   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1782 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1783 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1784 \def\bbl@usehooks#1#2{%
1785   \def\bbl@elth##1{%
1786     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@}{#2}}%
1787     \bbl@cs{ev@#1@}%
1788     \ifx\language\undefined\else % Test required for Plain (?)
1789       \def\bbl@elth##1{%
1790         \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1}{#2}}%
1791         \bbl@cl{ev@#1}%
1792       \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).

```

1793 \def\bbl@evargs{% <- don't delete this comma
1794   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1795   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1796   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1797   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1798   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 `\fontenc` 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.

```

1799 \bbl@trace{Defining babelensure}
1800 \newcommand\babelensure[2][]{% TODO - revise test files
1801   \AddBabelHook{babel-ensure}{afterextras}{%
1802     \ifcase\bbl@select@type
1803       \bbl@cl{e}%
1804     \fi}%
1805   \begingroup
1806     \let\bbl@ens@include\@empty
1807     \let\bbl@ens@exclude\@empty
1808     \def\bbl@ens@fontenc{\relax}%
1809     \def\bbl@tempb##1{%
1810       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1811     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1812     \def\bbl@tempb##1=##2\@{ \@namedef{bbl@ens@##1}{##2}}%
1813     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1814     \def\bbl@tempc{\bbl@ensure}%
1815     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1816       \expandafter{\bbl@ens@include}%
1817       \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%

```

```

1818 \expandafter{\bbl@ens@exclude}}%
1819 \toks@\expandafter{\bbl@tempc}%
1820 \bbl@exp{%
1821 \endgroup
1822 \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1823 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1824 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1825 \ifx##1\undefined % 3.32 - Don't assume the macro exists
1826 \edef##1{\noexpand\bbl@nocaption
1827 {\bbl@stripslash##1}{\language\language\bbl@stripslash##1}}%
1828 \fi
1829 \ifx##1\empty\else
1830 \in@{##1}{#2}%
1831 \ifin\else
1832 \bbl@ifunset{\bbl@ensure@\language\language}%
1833 {\bbl@exp{%
1834 \\\DeclareRobustCommand\<bbl@ensure@\language\language>[1]{%
1835 \\\foreignlanguage{\language\language}%
1836 {\ifx\relax#3\else
1837 \\\fontencoding{#3}\selectfont
1838 \fi
1839 #####1}}}%
1840 {}}%
1841 \toks@\expandafter{##1}%
1842 \edef##1{%
1843 \bbl@csarg\noexpand{\ensure@\language\language}%
1844 {\the\toks@}}}%
1845 \fi
1846 \expandafter\bbl@tempb
1847 \fi}%
1848 \expandafter\bbl@tempb\bbl@captionslist\today\empty
1849 \def\bbl@tempa##1{% elt for include list
1850 \ifx##1\empty\else
1851 \bbl@csarg\in@{\ensure@\language\language\expandafter}\expandafter{##1}%
1852 \ifin\else
1853 \bbl@tempb##1\empty
1854 \fi
1855 \expandafter\bbl@tempa
1856 \fi}%
1857 \bbl@tempa#1\empty}
1858 \def\bbl@captionslist{%
1859 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1860 \contentsname\listfigurename\listtablename\indexname\figurename
1861 \tablename\partname\enc1name\ccname\headtoname\pagename\seename
1862 \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`.

```

1863 \bbl@trace{Macros for setting language files up}
1864 \def\bbl@ldfinit{%
1865   \let\bbl@screset\@empty
1866   \let\BabelStrings\bbl@opt@string
1867   \let\BabelOptions\@empty
1868   \let\BabelLanguages\relax
1869   \ifx\originalTeX\@undefined
1870     \let\originalTeX\@empty
1871   \else
1872     \originalTeX
1873   \fi}
1874 \def\LdfInit#1#2{%
1875   \chardef\atcatcode=\catcode`\@
1876   \catcode`\@=11\relax
1877   \chardef\eqcatcode=\catcode`\=
1878   \catcode`\==12\relax
1879   \expandafter\if\expandafter\@backslashchar
1880     \expandafter\@car\string#2\@nil
1881   \ifx#2\@undefined\else
1882     \ldf@quit{#1}%
1883   \fi
1884 \else
1885   \expandafter\ifx\csname#2\endcsname\relax\else
1886     \ldf@quit{#1}%
1887   \fi
1888 \fi
1889 \bbl@ldfinit}

```

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

```

1890 \def\ldf@quit#1{%
1891   \expandafter\main@language\expandafter{#1}%
1892   \catcode`\@=\atcatcode \let\atcatcode\relax
1893   \catcode`\==\eqcatcode \let\eqcatcode\relax
1894   \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.

```

1895 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1896   \bbl@afterlang
1897   \let\bbl@afterlang\relax
1898   \let\BabelModifiers\relax
1899   \let\bbl@screset\relax}%
1900 \def\ldf@finish#1{%
1901   \ifx\loadlocalcfg\@undefined\else % For LaTeX 209
1902     \loadlocalcfg{#1}%
1903   \fi
1904   \bbl@afterldf{#1}%
1905   \expandafter\main@language\expandafter{#1}%

```

```

1906 \catcode\@=\atcatcode \let\atcatcode\relax
1907 \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  $\LaTeX$ .

```

1908 \@onlypreamble\LdfInit
1909 \@onlypreamble\ldf@quit
1910 \@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.

```

1911 \def\main@language#1{%
1912 \def\bbl@main@language{#1}%
1913 \let\language\main@language % TODO. Set localename
1914 \bbl@id@assign
1915 \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`.

```

1916 \def\bbl@beforestart{%
1917 \bbl@usehooks{beforestart}{}%
1918 \global\let\bbl@beforestart\relax}
1919 \AtBeginDocument{%
1920 \@nameuse{bbl@beforestart}%
1921 \if@filesw
1922 \providecommand\babel@aux[2]{}%
1923 \immediate\write\@mainaux{%
1924 \string\providecommand\string\babel@aux[2]{}%
1925 \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1926 \fi
1927 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1928 \ifbbl@single % must go after the line above.
1929 \renewcommand\selectlanguage[1]{}%
1930 \renewcommand\foreignlanguage[2]{#2}%
1931 \global\let\babel@aux\@gobbles % Also as flag
1932 \fi
1933 \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

```

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

```

1934 \def\select@language@x#1{%
1935 \ifcase\bbl@select@type
1936 \bbl@ifsamestring\language{#1}{\select@language{#1}}%
1937 \else
1938 \select@language{#1}%
1939 \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.

```

1940 \bbl@trace{Shorhands}
1941 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1942 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.

```

```

1943 \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\@makeother#1}}%
1944 \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1945 \begingroup
1946 \catcode`#1\active
1947 \nfss@catcodes
1948 \ifnum\catcode`#1=\active
1949 \endgroup
1950 \bbl@add\nfss@catcodes{\@makeother#1}%
1951 \else
1952 \endgroup
1953 \fi
1954 \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.

```

1955 \def\bbl@remove@special#1{%
1956 \begingroup
1957 \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1958 \else\noexpand##1\noexpand##2\fi}%
1959 \def\do{\x\do}%
1960 \def\@makeother{\x\@makeother}%
1961 \edef\x{\endgroup
1962 \def\noexpand\dospecials{\dospecials}%
1963 \expandafter\ifx\csname @sanitize\endcsname\relax\else
1964 \def\noexpand\@sanitize{\@sanitize}%
1965 \fi}%
1966 \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).

```

1967 \def\bbl@active@def#1#2#3#4{%
1968 \@namedef{#3#1}{%
1969 \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1970 \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1971 \else
1972 \bbl@afterfi\csname#2@sh@#1\endcsname
1973 \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.

```

1974 \long\@namedef{#3@arg#1}##1{%
1975 \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
1976 \bbl@afterelse\csname#4#1\endcsname##1%

```

```

1977 \else
1978 \bbl@afterfi\csname#2@sh@#1@\string##1@endcsname
1979 \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.

```

1980 \def\initiate@active@char#1{%
1981 \bbl@ifunset{active@char\string#1}%
1982 {\bbl@withactive
1983 {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1984 {}}

```

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).

```

1985 \def\@initiate@active@char#1#2#3{%
1986 \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1987 \ifx#1\@undefined
1988 \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
1989 \else
1990 \bbl@csarg\let{oridef@#2}#1%
1991 \bbl@csarg\edef{oridef@#2}{%
1992 \let\noexpand#1%
1993 \expandafter\noexpand\csname bbl@oridef@@#2@endcsname}%
1994 \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*").

```

1995 \ifx#1#3\relax
1996 \expandafter\let\csname normal@char#2@endcsname#3%
1997 \else
1998 \bbl@info{Making #2 an active character}%
1999 \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
2000 \@namedef{normal@char#2}{%
2001 \textormath{#3}{\csname bbl@oridef@@#2@endcsname}}%
2002 \else
2003 \@namedef{normal@char#2}{#3}%
2004 \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).

```

2005 \bbl@restoreactive{#2}%
2006 \AtBeginDocument{%
2007 \catcode`#2\active
2008 \if@filesw
2009 \immediate\write\@mainaux{\catcode`\string#2\active}%
2010 \fi}%
2011 \expandafter\bbl@add@special\csname#2@endcsname
2012 \catcode`#2\active
2013 \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⟩).

```

2014 \let\bbl@tempa\@firstoftwo
2015 \if\string^#2%
2016 \def\bbl@tempa{\noexpand\textormath}%
2017 \else
2018 \ifx\bbl@mathnormal\@undefined\else
2019 \let\bbl@tempa\bbl@mathnormal
2020 \fi
2021 \fi
2022 \expandafter\edef\csname active@char#2\endcsname{%
2023 \bbl@tempa
2024 {\noexpand\if@safe@actives
2025 \noexpand\expandafter
2026 \expandafter\noexpand\csname normal@char#2\endcsname
2027 \noexpand\else
2028 \noexpand\expandafter
2029 \expandafter\noexpand\csname bbl@doactive#2\endcsname
2030 \noexpand\fi}%
2031 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
2032 \bbl@csarg\edef{doactive#2}{%
2033 \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!).

```

2034 \bbl@csarg\edef{active@#2}{%
2035 \noexpand\active@prefix\noexpand#1%
2036 \expandafter\noexpand\csname active@char#2\endcsname}%
2037 \bbl@csarg\edef{normal@#2}{%
2038 \noexpand\active@prefix\noexpand#1%
2039 \expandafter\noexpand\csname normal@char#2\endcsname}%
2040 \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.

```

2041 \bbl@active@def#2\user@group{user@active}{language@active}%
2042 \bbl@active@def#2\language@group{language@active}{system@active}%
2043 \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.

```

2044 \expandafter\edef\csname\user@group @sh@#2@@\endcsname
2045 {\expandafter\noexpand\csname normal@char#2\endcsname}%
2046 \expandafter\edef\csname\user@group @sh@#2@string\protect\endcsname
2047 {\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.

```

2048 \if\string'#2%
2049 \let\prim@s\bbl@prim@s
2050 \let\active@math@prime#1%
2051 \fi
2052 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}

```

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

```

2053 <<{*More package options}>> ≡
2054 \DeclareOption{math=active}{}
2055 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
2056 <</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* the end of the ldf.

```

2057 \@ifpackagewith{babel}{KeepShorthandsActive}%
2058 {\let\bbl@restoreactive\@gobble}%
2059 {\def\bbl@restoreactive#1{%
2060 \bbl@exp{%
2061 \\\AfterBabelLanguage\\CurrentOption
2062 {\catcode`#1=\the\catcode`#1\relax}%
2063 \\\AtEndOfPackage
2064 {\catcode`#1=\the\catcode`#1\relax}}}%
2065 \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.

```

2066 \def\bbl@sh@select#1#2{%
2067 \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
2068 \bbl@afterelse\bbl@scndcs
2069 \else
2070 \bbl@afterfi\csname#1@sh@#2@sel\endcsname
2071 \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 `\protect`s 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.

```

2072 \begingroup
2073 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
2074 {\gdef\active@prefix#1{%
2075 \ifx\protect\@typeset@protect
2076 \else
2077 \ifx\protect\@unexpandable@protect
2078 \noexpand#1%
2079 \else
2080 \protect#1%
2081 \fi
2082 \expandafter\@gobble
2083 \fi}}
2084 {\gdef\active@prefix#1{%
2085 \ifincsname
2086 \string#1%
2087 \expandafter\@gobble

```



```

2088     \else
2089     \ifx\protect\@typeset@protect
2090     \else
2091     \ifx\protect\@unexpandable@protect
2092     \noexpand#1%
2093     \else
2094     \protect#1%
2095     \fi
2096     \expandafter\expandafter\expandafter\@gobble
2097     \fi
2098     \fi}}
2099 \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>`.

```

2100 \newif\if@safe@actives
2101 \@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.

```

2102 \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`.

```

2103 \def\bbl@activate#1{%
2104   \bbl@withactive{\expandafter\let\expandafter}#1%
2105   \csname bbl@active@\string#1\endcsname}
2106 \def\bbl@deactivate#1{%
2107   \bbl@withactive{\expandafter\let\expandafter}#1%
2108   \csname bbl@normal@\string#1\endcsname}

```

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

```

\bbl@scndcs
2109 \def\bbl@firstcs#1#2{\csname#1\endcsname}
2110 \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.

The auxiliary macro `\babel@texpdf` improves the interoperativity with `hyperref` and takes 4 arguments: (1) The  $\TeX$  code in text mode, (2) the string for `hyperref`, (3) the  $\TeX$  code in math mode, and (4), which is currently ignored, but it’s meant for a string in math mode, like a minus sign instead of an hyphen (currently `hyperref` doesn’t discriminate the mode). This macro may be used in `ldf` files.

```

2111 \def\babel@texpdf#1#2#3#4{%
2112   \ifx\texorpdfstring\undefined
2113   \textormath{#1}{#2}%
2114   \else
2115   \texorpdfstring{\textormath{#1}{#3}}{#2}%
2116   % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
2117   \fi}
2118 %

```

```

2119 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
2120 \def\@decl@short#1#2#3\@nil#4{%
2121   \def\bbl@tempa{#3}%
2122   \ifx\bbl@tempa\@empty
2123     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
2124     \bbl@ifunset{#1@sh@\string#2@}{}%
2125     {\def\bbl@tempa{#4}%
2126       \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
2127       \else
2128         \bbl@info
2129           {Redefining #1 shorthand \string#2\%
2130           in language \CurrentOption}%
2131       \fi}%
2132     \@namedef{#1@sh@\string#2@}{#4}%
2133   \else
2134     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
2135     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
2136     {\def\bbl@tempa{#4}%
2137       \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
2138       \else
2139         \bbl@info
2140           {Redefining #1 shorthand \string#2\string#3\%
2141           in language \CurrentOption}%
2142       \fi}%
2143     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2144   \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.

```

2145 \def\textormath{%
2146   \ifmmode
2147     \expandafter\@secondoftwo
2148   \else
2149     \expandafter\@firstoftwo
2150   \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’.

```

2151 \def\user@group{user}
2152 \def\language@group{english} % TODO. I don't like defaults
2153 \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.

```

2154 \def\useshorthands{%
2155   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}
2156   \def\bbl@usesh@s#1{%
2157     \bbl@usesh@x
2158     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
2159     {#1}}
2160 \def\bbl@usesh@x#1#2{%
2161   \bbl@ifshorthand{#2}%
2162   {\def\user@group{user}%
2163     \initiate@active@char{#2}%
2164     #1%
2165     \bbl@activate{#2}}%

```

```

2166     {\bbl@error
2167       {Cannot declare a shorthand turned off (\string#2)}}
2168     {Sorry, but you cannot use shorthands which have been\\%
2169       turned off in the package options}}}}

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

2170 \def\user@language@group{user@\language@group}
2171 \def\bbl@set@user@generic#1#2{%
2172   \bbl@ifunset{user@generic@active#1}%
2173   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
2174     \bbl@active@def#1\user@group{user@generic@active}{language@active}%
2175     \expandafter\edef\csname#2@sh@#1@\endcsname{%
2176       \expandafter\noexpand\csname normal@char#1\endcsname}%
2177     \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%
2178       \expandafter\noexpand\csname user@active#1\endcsname}}%
2179   \@empty}
2180 \newcommand\defineshorthand[3][user]{%
2181   \edef\bbl@tempa{\zap@space#1 \@empty}%
2182   \bbl@for\bbl@tempb\bbl@tempa{%
2183     \if*\expandafter\@car\bbl@tempb\@nil
2184       \edef\bbl@tempb{user\expandafter@gobble\bbl@tempb}%
2185       \@expandtwoargs
2186       \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
2187     \fi
2188     \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 to fix it in the same way languages names are fixed. [TODO].

2189 \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".

2190 \def\aliasshorthand#1#2{%
2191   \bbl@ifshorthand{#2}%
2192   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2193     \ifx\document\@notprerr
2194       \@notshorthand{#2}%
2195     \else
2196       \initiate@active@char{#2}%
2197       \expandafter\let\csname active@char\string#2\expandafter\endcsname
2198         \csname active@char\string#1\endcsname
2199       \expandafter\let\csname normal@char\string#2\expandafter\endcsname
2200         \csname normal@char\string#1\endcsname
2201       \bbl@activate{#2}%
2202     \fi
2203   \fi}%
2204   {\bbl@error
2205     {Cannot declare a shorthand turned off (\string#2)}}
2206   {Sorry, but you cannot use shorthands which have been\\%
2207     turned off in the package options}}}}

\@notshorthand

```

```

2208 \def\@notshorthand#1{%
2209   \bbl@error{%
2210     The character '\string #1' should be made a shorthand character;\%
2211     add the command \string\usesshorthands\string{#1\string} to
2212     the preamble.\%
2213     I will ignore your instruction}%
2214   {You may proceed, but expect unexpected results}}

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

2215 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2216 \DeclareRobustCommand*\shorthandoff{%
2217   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2218 \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.

2219 \def\bbl@switch@sh#1#2{%
2220   \ifx#2\@nnil\else
2221     \bbl@ifunset{\bbl@active@\string#2}%
2222     {\bbl@error
2223       {I cannot switch '\string#2' on or off--not a shorthand}%
2224       {This character is not a shorthand. Maybe you made\%
2225         a typing mistake? I will ignore your instruction}}}%
2226     {\ifcase#1%
2227       \catcode`#212\relax
2228       \or
2229       \catcode`#2\active
2230       \or
2231       \csname bbl@oricat@\string#2\endcsname
2232       \csname bbl@oridef@\string#2\endcsname
2233       \fi}%
2234     \bbl@afterfi\bbl@switch@sh#1%
2235   \fi}

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

2236 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
2237 \def\bbl@putsh#1{%
2238   \bbl@ifunset{\bbl@active@\string#1}%
2239   {\bbl@putsh@i#1\@empty\@nnil}%
2240   {\csname bbl@active@\string#1\endcsname}}
2241 \def\bbl@putsh@i#1#2\@nnil{%
2242   \csname\language@group @sh@\string#1@%
2243     \ifx\@empty#2\else\string#2@\fi\endcsname}
2244 \ifx\bbl@opt@shorthands\@nnil\else
2245   \let\bbl@s@initiate@active@char\initiate@active@char
2246   \def\initiate@active@char#1{%
2247     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
2248   \let\bbl@s@switch@sh\bbl@switch@sh
2249   \def\bbl@switch@sh#1#2{%
2250     \ifx#2\@nnil\else
2251       \bbl@afterfi
2252       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%

```

```

2253 \fi}
2254 \let\bbl@s@activate\bbl@activate
2255 \def\bbl@activate#1{%
2256 \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
2257 \let\bbl@s@deactivate\bbl@deactivate
2258 \def\bbl@deactivate#1{%
2259 \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
2260 \fi

```

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

```

2261 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{bbl@active@string#1}{#3}{#2}}

```

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

```

2262 \def\bbl@prim@s{%
2263 \prime\futurelet\@let@token\bbl@pr@m@s}
2264 \def\bbl@if@primes#1#2{%
2265 \ifx#1\@let@token
2266 \expandafter\@firstoftwo
2267 \else\ifx#2\@let@token
2268 \bbl@afterelse\expandafter\@firstoftwo
2269 \else
2270 \bbl@afterfi\expandafter\@secondoftwo
2271 \fi\fi}
2272 \begingroup
2273 \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
2274 \catcode`\'=12 \catcode`\"=\active \lccode`\"=\'
2275 \lowercase{%
2276 \gdef\bbl@pr@m@s{%
2277 \bbl@if@primes"%
2278 \pr@@@s
2279 {\bbl@if@primes*\^pr@@@t\egroup}}}
2280 \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).

```

2281 \initiate@active@char{~}
2282 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2283 \bbl@activate{~}

```

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

```

2284 \expandafter\def\csname OT1dqpos\endcsname{127}
2285 \expandafter\def\csname T1dqpos\endcsname{4}

```

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

```

2286 \ifx\f@encoding\undefined
2287 \def\f@encoding{OT1}
2288 \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.

```
2289 \bbl@trace{Language attributes}
2290 \newcommand\languageattribute[2]{%
2291   \def\bbl@tempc{#1}%
2292   \bbl@fixname\bbl@tempc
2293   \bbl@iflanguage\bbl@tempc{%
2294     \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.

```
2295     \ifx\bbl@known@attrs\@undefined
2296       \in@false
2297     \else
2298       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
2299     \fi
2300     \ifin@
2301       \bbl@warning{%
2302         You have more than once selected the attribute '##1'\%
2303         for language #1. Reported}%
2304     \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.

```
2305       \bbl@exp{%
2306         \\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
2307       \edef\bbl@tempa{\bbl@tempc-##1}%
2308       \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
2309       {\csname\bbl@tempc @attr##1\endcsname}%
2310       {\@attrerr{\bbl@tempc}{##1}}%
2311     \fi}}
2312 \@onlypreamble\languageattribute
```

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

```
2313 \newcommand*{\@attrerr}[2]{%
2314   \bbl@error
2315   {The attribute #2 is unknown for language #1.}%
2316   {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}`.

```
2317 \def\bbl@declare@ttribute#1#2#3{%
2318   \bbl@xin@{,#2,}{,\BabelModifiers,}%
2319   \ifin@
2320     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2321   \fi
2322   \bbl@add@list\bbl@attributes{#1-#2}%
2323   \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`'.

```
2324 \def\bbl@ifattributeset#1#2#3#4{%
2325   \ifx\bbl@known@attribs\@undefined
2326     \in@false
2327   \else
2328     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
2329   \fi
2330   \ifin@
2331     \bbl@afterelse#3%
2332   \else
2333     \bbl@afterfi#4%
2334   \fi
2335 }
```

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

We first assume the attribute is unknown. Then we loop over the list of known attributes, trying to find a match. When a match is found the definition of `\bbl@tempa` is changed. Finally we execute `\bbl@tempa`.

```
2336 \def\bbl@ifknown@ttrib#1#2{%
2337   \let\bbl@tempa\@secondoftwo
2338   \bbl@loopx\bbl@tempb{#2}{%
2339     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
2340   \ifin@
2341     \let\bbl@tempa\@firstoftwo
2342   \else
2343     \fi}%
2344   \bbl@tempa
2345 }
```

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

```
2346 \def\bbl@clear@ttribs{%
2347   \ifx\bbl@attributes\@undefined\else
2348     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
2349       \expandafter\bbl@clear@ttrib\bbl@tempa.
2350     }%
2351   \let\bbl@attributes\@undefined
2352   \fi}
2353 \def\bbl@clear@ttrib#1-#2.{%
2354   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
2355 \AtBeginDocument{\bbl@clear@ttribs}
```

## 9.7 Support for saving macro definitions

To save the meaning of control sequences using `\babel@save`, we use temporary control sequences. To save hash table entries for these control sequences, we don't use the name of the control sequence to be saved to construct the temporary name. Instead we simply use the value of a counter, which is reset to zero each time we begin to save new values. This works well because we release the saved meanings before we begin to save a new set of control sequence meanings (see `\selectlanguage`

and `\originalTeX`). Note undefined macros are not undefined any more when saved – they are `\relax`'ed.

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

```
2356 \bbl@trace{Macros for saving definitions}
2357 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
2358 \newcount\babel@savecnt
2359 \babel@beginsave
```

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

```
2360 \def\babel@save#1{%
2361   \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
2362   \toks@\expandafter{\originalTeX\let#1=}%
2363   \bbl@exp{%
2364     \def\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
2365     \advance\babel@savecnt\@ne}
2366 \def\babel@savevariable#1{%
2367   \toks@\expandafter{\originalTeX #1=}%
2368   \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@nonfrenchspacing` switches it on when it isn't already in effect and `\bbl@nonfrenchspacing` switches it off if necessary.

```
2369 \def\bbl@frenchspacing{%
2370   \ifnum\the\sfcode\.=\@m
2371     \let\bbl@nonfrenchspacing\relax
2372   \else
2373     \frenchspacing
2374     \let\bbl@nonfrenchspacing\nonfrenchspacing
2375   \fi}
2376 \let\bbl@nonfrenchspacing\nonfrenchspacing
2377 %
2378 \let\bbl@elt\relax
2379 \edef\bbl@fs@chars{%
2380   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
2381   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
2382   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
```

## 9.8 Short tags

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

```
2383 \bbl@trace{Short tags}
2384 \def\babeltags#1{%
2385   \edef\bbl@tempa{\zap@space#1 \@empty}%
2386   \def\bbl@tempb##1=##2\@{#}%
2387   \edef\bbl@tempc{%
2388     \noexpand\newcommand
2389     \expandafter\noexpand\csname ##1\endcsname{%
```

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



```

2390 \noexpand\protect
2391 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
2392 \noexpand\newcommand
2393 \expandafter\noexpand\csname text##1\endcsname{%
2394 \noexpand\foreignlanguage{##2}}}%
2395 \bbl@tempc}%
2396 \bbl@for\bbl@tempa\bbl@tempa{%
2397 \expandafter\bbl@tempb\bbl@tempa\@@}}

```

## 9.9 Hyphens

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

```

2398 \bbl@trace{Hyphens}
2399 \@onlypreamble\babelhyphenation
2400 \AtEndOfPackage{%
2401 \newcommand\babelhyphenation[2][\@empty]{%
2402 \ifx\bbl@hyphenation@relax
2403 \let\bbl@hyphenation@\@empty
2404 \fi
2405 \ifx\bbl@hyphlist\@empty\else
2406 \bbl@warning{%
2407 You must not intermingle \string\selectlanguage\space and\\%
2408 \string\babelhyphenation\space or some exceptions will not\\%
2409 be taken into account. Reported}%
2410 \fi
2411 \ifx\@empty#1%
2412 \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2413 \else
2414 \bbl@vforeach{#1}{%
2415 \def\bbl@tempa{##1}%
2416 \bbl@fixname\bbl@tempa
2417 \bbl@iflanguage\bbl@tempa{%
2418 \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2419 \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
2420 \@empty
2421 {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
2422 #2}}}%
2423 \fi}}

```

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

```

2424 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\zskip\fi}
2425 \def\bbl@t@one{T1}
2426 \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.

```

2427 \newcommand\babellnullhyphen{\char\hyphenchar\font}
2428 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
2429 \def\bbl@hyphen{%
2430 \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
2431 \def\bbl@hyphen@i#1#2{%
2432 \bbl@ifunset{bbl@hy@#1#2\@empty}%

```

<sup>32</sup> $\TeX$  begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

```

2433 {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
2434 {\csname bbl@hy@#1#2\@empty\endcsname}}

```

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

There should not be a discretionary after a hyphen at the beginning of a word, so it is prevented if preceded by a skip. Unfortunately, this does handle cases like “(-suffix)”. \nobreak is always preceded by \leavevmode, in case the shorthand starts a paragraph.

```

2435 \def\bbl@usehyphen#1{%
2436   \leavevmode
2437   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2438   \nobreak\hskip\z@skip}
2439 \def\bbl@@usehyphen#1{%
2440   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

2441 \def\bbl@hyphenchar{%
2442   \ifnum\hyphenchar\font=\m@ne
2443     \babeinullhyphen
2444   \else
2445     \char\hyphenchar\font
2446   \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.

```

2447 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2448 \def\bbl@hy@@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2449 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2450 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
2451 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
2452 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
2453 \def\bbl@hy@repeat{%
2454   \bbl@usehyphen{%
2455     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2456 \def\bbl@hy@repeat{%
2457   \bbl@usehyphen{%
2458     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2459 \def\bbl@hy@empty{\hskip\z@skip}
2460 \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.

```

2461 \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.

```

2462 \bbl@trace{Multiencoding strings}
2463 \def\bbl@tglobal#1{\global\let#1#1}
2464 \def\bbl@recatcode#1{% TODO. Used only once?
2465   \@tempcnta="7F

```

```

2466 \def\bb1@tempa{%
2467   \ifnum\@tempcnta>"FF\else
2468     \catcode\@tempcnta=#1\relax
2469     \advance\@tempcnta@ne
2470     \expandafter\bb1@tempa
2471   \fi}%
2472 \bb1@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 `\bb1@uclc`. The parser is restarted inside `\lang@bb1@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\bb1@tolower\@empty\bb1@toupper\@empty
```

and starts over (and similarly when lowercasing).

```

2473 \@ifpackagewith{babel}{nocase}%
2474   {\let\bb1@patchuclc\relax}%
2475   {\def\bb1@patchuclc{%
2476     \global\let\bb1@patchuclc\relax
2477     \g@addto@macro\uclclist{\reserved@b{\reserved@b\bb1@uclc}}%
2478     \gdef\bb1@uclc##1{%
2479       \let\bb1@encoded\bb1@encoded@uclc
2480       \bb1@ifunset{\language @bb1@uclc}% and resumes it
2481       {##1}%
2482       {\let\bb1@tempa##1\relax % Used by LANG@bb1@uclc
2483         \csname\language @bb1@uclc\endcsname}%
2484       {\bb1@tolower\@empty}{\bb1@toupper\@empty}}%
2485     \gdef\bb1@tolower{\csname\language @bb1@lc\endcsname}%
2486     \gdef\bb1@toupper{\csname\language @bb1@uc\endcsname}}%
2487 <<More package options>> ≡
2488 \DeclareOption{nocase}{}
2489 <</More package options>>

```

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

```

2490 <<More package options>> ≡
2491 \let\bb1@opt@strings\@nnil % accept strings=value
2492 \DeclareOption{strings}{\def\bb1@opt@strings{\BabelStringsDefault}}
2493 \DeclareOption{strings=encoded}{\let\bb1@opt@strings\relax}
2494 \def\BabelStringsDefault{generic}
2495 <</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.

```

2496 \@onlypreamble\StartBabelCommands
2497 \def\StartBabelCommands{%
2498   \begingroup
2499   \bb1@recatcode{11}%
2500   <<Macros local to BabelCommands>>
2501   \def\bb1@provstring##1##2{%
2502     \providecommand##1{##2}%
2503     \bb1@tglobal##1}%
2504   \global\let\bb1@scafter\@empty

```

```

2505 \let\StartBabelCommands\bbl@startcmds
2506 \ifx\BabelLanguages\relax
2507   \let\BabelLanguages\CurrentOption
2508 \fi
2509 \begingroup
2510 \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2511 \StartBabelCommands}
2512 \def\bbl@startcmds{%
2513   \ifx\bbl@screset\@nnil\else
2514     \bbl@usehooks{stopcommands}{}%
2515   \fi
2516 \endgroup
2517 \begingroup
2518 \@ifstar
2519   {\ifx\bbl@opt@strings\@nnil
2520     \let\bbl@opt@strings\BabelStringsDefault
2521     \fi
2522     \bbl@startcmds@i}%
2523   \bbl@startcmds@i}
2524 \def\bbl@startcmds@i#1#2{%
2525   \edef\bbl@L{\zap@space#1 \@empty}%
2526   \edef\bbl@G{\zap@space#2 \@empty}%
2527   \bbl@startcmds@ii}
2528 \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.

```

2529 \newcommand\bbl@startcmds@ii[1][\@empty]{%
2530   \let\SetString@gobbletwo
2531   \let\bbl@stringdef@gobbletwo
2532   \let\AfterBabelCommands@gobble
2533   \ifx\@empty#1%
2534     \def\bbl@sc@label{generic}%
2535     \def\bbl@encstring##1##2{%
2536       \ProvideTextCommandDefault##1{##2}%
2537       \bbl@toglobal##1%
2538       \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
2539     \let\bbl@sc@test\in@true
2540   \else
2541     \let\bbl@sc@charset\space % <- zapped below
2542     \let\bbl@sc@fontenc\space % <- " "
2543     \def\bbl@tempa##1=##2\@nil{%
2544       \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
2545     \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
2546     \def\bbl@tempa##1 ##2{% space -> comma
2547       ##1%
2548       \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
2549     \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
2550     \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
2551     \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
2552     \def\bbl@encstring##1##2{%
2553       \bbl@foreach\bbl@sc@fontenc{%

```

```

2554 \bbl@ifunset{T@####1}%
2555 {}%
2556 {\ProvideTextCommand##1{####1}{##2}%
2557 \bbl@tglobal##1%
2558 \expandafter
2559 \bbl@tglobal\csname####1\string##1\endcsname}}}%
2560 \def\bbl@sctest{%
2561 \bbl@xin@{,\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
2562 \fi
2563 \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
2564 \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
2565 \let\AfterBabelCommands\bbl@aftercmds
2566 \let\SetString\bbl@setstring
2567 \let\bbl@stringdef\bbl@encstring
2568 \else % ie, strings=value
2569 \bbl@sctest
2570 \ifin@
2571 \let\AfterBabelCommands\bbl@aftercmds
2572 \let\SetString\bbl@setstring
2573 \let\bbl@stringdef\bbl@provstring
2574 \fi\fi\fi
2575 \bbl@scswitch
2576 \ifx\bbl@G\@empty
2577 \def\SetString##1##2{%
2578 \bbl@error{Missing group for string \string##1}%
2579 {You must assign strings to some category, typically\\%
2580 captions or extras, but you set none}}%
2581 \fi
2582 \ifx\@empty#1%
2583 \bbl@usehooks{defaultcommands}}}%
2584 \else
2585 \@expandtwoargs
2586 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}}%
2587 \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).

```

2588 \def\bbl@forlang#1#2{%
2589 \bbl@for#1\bbl@L{%
2590 \bbl@xin@{,#1,}{,\BabelLanguages,}%
2591 \ifin@#2\relax\fi}}
2592 \def\bbl@scswitch{%
2593 \bbl@forlang\bbl@tempa{%
2594 \ifx\bbl@G\@empty\else
2595 \ifx\SetString\@gobbletwo\else
2596 \edef\bbl@GL{\bbl@G\bbl@tempa}%
2597 \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
2598 \ifin@\else
2599 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
2600 \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
2601 \fi
2602 \fi
2603 \fi}}

```

```

2604 \AtEndOfPackage{%
2605   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
2606   \let\bbl@scswitch\relax}
2607 \@onlypreamble\EndBabelCommands
2608 \def\EndBabelCommands{%
2609   \bbl@usehooks{stopcommands}{}%
2610   \endgroup
2611   \endgroup
2612   \bbl@scafter}
2613 \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.

```

2614 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
2615   \bbl@forlang\bbl@tempa{%
2616     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2617     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2618       {\bbl@exp{%
2619         \global\bbbl@add\<\bbl@G\bbl@tempa>{\bbbl@scset\#1\<\bbl@LC>}}}%
2620       }%
2621     \def\BabelString{#2}%
2622     \bbl@usehooks{stringprocess}{}%
2623     \expandafter\bbl@stringdef
2624     \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.

```

2625 \ifx\bbl@opt@strings\relax
2626   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2627   \bbl@patchuclc
2628   \let\bbl@encoded\relax
2629   \def\bbl@encoded@uclc#1{%
2630     \@inmathwarn#1%
2631     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2632       \expandafter\ifx\csname ?\string#1\endcsname\relax
2633         \TextSymbolUnavailable#1%
2634       \else
2635         \csname ?\string#1\endcsname
2636       \fi
2637     \else
2638       \csname\cf@encoding\string#1\endcsname
2639     \fi}
2640 \else
2641   \def\bbl@scset#1#2{\def#1{#2}}
2642 \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.

```

2643 << *Macros local to BabelCommands >> ≡
2644 \def\SetStringLoop##1##2{%
2645   \def\bbl@temp1####1{\expandafter\noexpand\csname##1\endcsname}%
2646   \count@\z@

```

```

2647 \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2648 \advance\count@ \ne
2649 \toks@\expandafter{\bbl@tempa}%
2650 \bbl@exp{%
2651 \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2652 \count@=\the\count@\relax}}}%
2653 <</Macros local to BabelCommands>>

```

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

```

2654 \def\bbl@aftercmds#1{%
2655 \toks@\expandafter{\bbl@scafter#1}%
2656 \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.

```

2657 <<(*Macros local to BabelCommands)>> ≡
2658 \newcommand\SetCase[3][]{%
2659 \bbl@patchuclc
2660 \bbl@forlang\bbl@tempa{%
2661 \expandafter\bbl@encstring
2662 \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
2663 \expandafter\bbl@encstring
2664 \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2665 \expandafter\bbl@encstring
2666 \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2667 <</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.

```

2668 <<(*Macros local to BabelCommands)>> ≡
2669 \newcommand\SetHyphenMap[1]{%
2670 \bbl@forlang\bbl@tempa{%
2671 \expandafter\bbl@stringdef
2672 \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
2673 <</Macros local to BabelCommands>>

```

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

```

2674 \newcommand\BabelLower[2]{% one to one.
2675 \ifnum\lccode#1=#2\else
2676 \babel@savevariable{\lccode#1}%
2677 \lccode#1=#2\relax
2678 \fi}
2679 \newcommand\BabelLowerMM[4]{% many-to-many
2680 \@tempcnta=#1\relax
2681 \@tempcntb=#4\relax
2682 \def\bbl@tempa{%
2683 \ifnum\@tempcnta>#2\else
2684 \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2685 \advance\@tempcnta#3\relax
2686 \advance\@tempcntb#3\relax
2687 \expandafter\bbl@tempa
2688 \fi}%
2689 \bbl@tempa}
2690 \newcommand\BabelLowerMO[4]{% many-to-one
2691 \@tempcnta=#1\relax
2692 \def\bbl@tempa{%

```

```

2693 \ifnum\@tempcnta>#2\else
2694 \expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
2695 \advance\@tempcnta#3
2696 \expandafter\bb1@tempa
2697 \fi}%
2698 \bb1@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

2699 <<{*More package options}>> ≡
2700 \DeclareOption{hyphenmap=off}{\chardef\bb1@opt@hyphenmap\z@}
2701 \DeclareOption{hyphenmap=first}{\chardef\bb1@opt@hyphenmap\@ne}
2702 \DeclareOption{hyphenmap=select}{\chardef\bb1@opt@hyphenmap\tw@}
2703 \DeclareOption{hyphenmap=other}{\chardef\bb1@opt@hyphenmap\thr@@}
2704 \DeclareOption{hyphenmap=other*}{\chardef\bb1@opt@hyphenmap4\relax}
2705 <</More package options>>

```

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

```

2706 \AtEndOfPackage{%
2707 \ifx\bb1@opt@hyphenmap\undefined
2708 \bb1@xin@{,}{\bb1@language@opts}%
2709 \chardef\bb1@opt@hyphenmap\ifin@4\else\@ne\fi
2710 \fi}

```

This section ends with a general tool for resetting the caption names with a unique interface. With the old way, which mixes the switcher and the string, we convert it to the new one, which separates these two steps.

```

2711 \newcommand\setlocalecaption{% TODO. Catch typos. What about ensure?
2712 \@ifstar\bb1@setcaption@s\bb1@setcaption@x}
2713 \def\bb1@setcaption@x#1#2#3{% language caption-name string
2714 \bb1@trim@def\bb1@tempa{#2}%
2715 \bb1@xin@{.template}{\bb1@tempa}%
2716 \ifin@
2717 \bb1@ini@captions@template{#3}{#1}%
2718 \else
2719 \edef\bb1@tempd{%
2720 \expandafter\expandafter\expandafter
2721 \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2722 \bb1@xin@
2723 {\expandafter\string\csname #2name\endcsname}%
2724 {\bb1@tempd}%
2725 \ifin@ % Renew caption
2726 \bb1@xin@{\string\bb1@scset}{\bb1@tempd}%
2727 \ifin@
2728 \bb1@exp{%
2729 \bb1@ifsamestring{\bb1@tempa}{\language@name}%
2730 {\bb1@scset\<#2name>\<#1#2name>}%
2731 {}}%
2732 \else % Old way converts to new way
2733 \bb1@ifunset{#1#2name}%
2734 {\bb1@exp{%
2735 \bb1@add\<captions#1>\<def\<#2name>\<#1#2name>}}%
2736 \bb1@ifsamestring{\bb1@tempa}{\language@name}%
2737 {\def\<#2name>\<#1#2name>}}%
2738 {}}}%
2739 {}%
2740 \fi
2741 \else
2742 \bb1@xin@{\string\bb1@scset}{\bb1@tempd}% New
2743 \ifin@ % New way

```



```

2744 \bbl@exp{%
2745 \\\bbl@add\<captions#1>\{\bbl@scset\<#2name>\<#1#2name>\}%
2746 \\\bbl@ifsamestring{\bbl@tempa}{\language\name}%
2747 {\bbl@scset\<#2name>\<#1#2name>\}%
2748 }%
2749 \else % Old way, but defined in the new way
2750 \bbl@exp{%
2751 \\\bbl@add\<captions#1>\{\def\<#2name>\{\<#1#2name>\}}%
2752 \\\bbl@ifsamestring{\bbl@tempa}{\language\name}%
2753 {\def\<#2name>\{\<#1#2name>\}}%
2754 }%
2755 \fi%
2756 \fi
2757 \@namedef{#1#2name}{#3}%
2758 \toks@\expandafter{\bbl@captionslist}%
2759 \bbl@exp{\in@{\<#2name>}{\the\toks@}}%
2760 \ifin@
2761 \bbl@exp{\bbl@add\{\bbl@captionslist{\<#2name>}}%
2762 \bbl@tglobal\bbl@captionslist
2763 \fi
2764 \fi}
2765 % \def\bbl@setcaption@s#1#2#3{} % TODO. Not yet implemented

```

## 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.

```

2766 \bbl@trace{Macros related to glyphs}
2767 \def\set@low@box#1{\setbox\tw@hbox{,}\setbox\z@hbox{#1}%
2768 \dimen\z@ht\z@ \advance\dimen\z@ -\ht\tw@%
2769 \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.

```

2770 \def\save@sf@q#1{\leavevmode
2771 \begingroup
2772 \edef\SF{\spacefactor\the\spacefactor}#1\SF
2773 \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.

```

2774 \ProvideTextCommand{\quotedblbase}{OT1}{%
2775 \save@sf@q{\set@low@box{\textquotedblright\}%
2776 \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.

```

2777 \ProvideTextCommandDefault{\quotedblbase}{%
2778 \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```
2779 \ProvideTextCommand{\quotesinglbase}{OT1}{%  
2780   \save@sf@q{\set@low@box{\textquoteright\}%  
2781     \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.

```
2782 \ProvideTextCommandDefault{\quotesinglbase}{%  
2783   \UseTextSymbol{OT1}{\quotesinglbase}}
```

`\guillemetleft` The guillemet characters are not available in OT1 encoding. They are faked. (Wrong names with o  
`\guillemetright` preserved for compatibility.)

```
2784 \ProvideTextCommand{\guillemetleft}{OT1}{%  
2785   \ifmmode  
2786     \ll  
2787   \else  
2788     \save@sf@q{\nobreak  
2789       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%  
2790   \fi}  
2791 \ProvideTextCommand{\guillemetright}{OT1}{%  
2792   \ifmmode  
2793     \gg  
2794   \else  
2795     \save@sf@q{\nobreak  
2796       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%  
2797   \fi}  
2798 \ProvideTextCommand{\guillemotleft}{OT1}{%  
2799   \ifmmode  
2800     \ll  
2801   \else  
2802     \save@sf@q{\nobreak  
2803       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%  
2804   \fi}  
2805 \ProvideTextCommand{\guillemotright}{OT1}{%  
2806   \ifmmode  
2807     \gg  
2808   \else  
2809     \save@sf@q{\nobreak  
2810       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%  
2811   \fi}
```

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

```
2812 \ProvideTextCommandDefault{\guillemetleft}{%  
2813   \UseTextSymbol{OT1}{\guillemetleft}}  
2814 \ProvideTextCommandDefault{\guillemetright}{%  
2815   \UseTextSymbol{OT1}{\guillemetright}}  
2816 \ProvideTextCommandDefault{\guillemotleft}{%  
2817   \UseTextSymbol{OT1}{\guillemotleft}}  
2818 \ProvideTextCommandDefault{\guillemotright}{%  
2819   \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
2820 \ProvideTextCommand{\guilsinglleft}{OT1}{%  
2821   \ifmmode  
2822     <%  
2823   \else  
2824     \save@sf@q{\nobreak  
2825       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%  
2826   \fi}
```

```

2827 \ProvideTextCommand{\guilsinglright}{OT1}{%
2828   \ifmmode
2829     >%
2830   \else
2831     \save@sf@q{\nobreak
2832       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2833   \fi}

```

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

```

2834 \ProvideTextCommandDefault{\guilsinglleft}{%
2835   \UseTextSymbol{OT1}{\guilsinglleft}}
2836 \ProvideTextCommandDefault{\guilsinglright}{%
2837   \UseTextSymbol{OT1}{\guilsinglright}}

```

### 9.12.2 Letters

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

```

2838 \DeclareTextCommand{\ij}{OT1}{%
2839   i\kern-0.02em\bbl@allowhyphens j}
2840 \DeclareTextCommand{\IJ}{OT1}{%
2841   I\kern-0.02em\bbl@allowhyphens J}
2842 \DeclareTextCommand{\ij}{T1}{\char188}
2843 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2844 \ProvideTextCommandDefault{\ij}{%
2845   \UseTextSymbol{OT1}{\ij}}
2846 \ProvideTextCommandDefault{\IJ}{%
2847   \UseTextSymbol{OT1}{\IJ}}

```

`\dj` The croatian language needs the letters `\dj` and `\DJ`; they are available in the T1 encoding, 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).

```

2848 \def\crrtic@{\hrule height0.1ex width0.3em}
2849 \def\crrtic@{\hrule height0.1ex width0.33em}
2850 \def\ddj@{%
2851   \setbox0\hbox{d}\dimen@=\ht0
2852   \advance\dimen@1ex
2853   \dimen@.45\dimen@
2854   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2855   \advance\dimen@ii.5ex
2856   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2857 \def\DDJ@{%
2858   \setbox0\hbox{D}\dimen@=.55\ht0
2859   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2860   \advance\dimen@ii.15ex % correction for the dash position
2861   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2862   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2863   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2864 %
2865 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2866 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2867 \ProvideTextCommandDefault{\dj}{%
2868   \UseTextSymbol{OT1}{\dj}}
2869 \ProvideTextCommandDefault{\DJ}{%
2870   \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.

```
2871 \DeclareTextCommand{\SS}{OT1}{SS}
2872 \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 2873 \ProvideTextCommandDefault{\glq}{%
2874 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2875 \ProvideTextCommand{\grq}{T1}{%
2876 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}%
2877 \ProvideTextCommand{\grq}{TU}{%
2878 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2879 \ProvideTextCommand{\grq}{OT1}{%
2880 \save@sf@q{\kern-.0125em
2881 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2882 \kern.07em\relax}}
2883 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}{\grq}}
```

`\glqq` The ‘german’ double quotes.

```
\grqq 2884 \ProvideTextCommandDefault{\glqq}{%
2885 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2886 \ProvideTextCommand{\grqq}{T1}{%
2887 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2888 \ProvideTextCommand{\grqq}{TU}{%
2889 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2890 \ProvideTextCommand{\grqq}{OT1}{%
2891 \save@sf@q{\kern-.07em
2892 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2893 \kern.07em\relax}}
2894 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}{\grqq}}
```

`\flq` The ‘french’ single guillemets.

```
\frq 2895 \ProvideTextCommandDefault{\flq}{%
2896 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}%
2897 \ProvideTextCommandDefault{\frq}{%
2898 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}%
2899 \ProvideTextCommandDefault{\flqq}{%
2900 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}%
2901 \ProvideTextCommandDefault{\frqq}{%
2902 \textormath{\guillemetright}{\mbox{\guillemetright}}}%
2903 \ProvideTextCommandDefault{\flq}{\UseTextSymbol{OT1}{\flq}}
2904 \ProvideTextCommandDefault{\frq}{\UseTextSymbol{OT1}{\frq}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq 2899 \ProvideTextCommandDefault{\flqq}{%
2900 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}%
2901 \ProvideTextCommandDefault{\frqq}{%
2902 \textormath{\guillemetright}{\mbox{\guillemetright}}}%
2903 \ProvideTextCommandDefault{\flq}{\UseTextSymbol{OT1}{\flq}}
2904 \ProvideTextCommandDefault{\frq}{\UseTextSymbol{OT1}{\frq}}
```

### 9.12.4 Umlauts and tremas

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

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

```
2903 \def\umlauthigh{%
2904   \def\bbl@umlauta##1{\leavevmode\bgroup%
2905     \expandafter\accent\csname\fontencoding dqpos\endcsname
2906     ##1\bbl@allowhyphens\egroup}%
2907   \let\bbl@umlaute\bbl@umlauta}
2908 \def\umlautlow{%
2909   \def\bbl@umlauta{\protect\lower@umlaut}}
2910 \def\umlautelow{%
2911   \def\bbl@umlaute{\protect\lower@umlaut}}
2912 \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.

```
2913 \expandafter\ifx\csname U@D\endcsname\relax
2914   \csname newdimen\endcsname\U@D
2915 \fi
```

The following code fools  $\TeX$ '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.

```
2916 \def\lower@umlaut#1{%
2917   \leavevmode\bgroup
2918     \U@D 1ex%
2919     {\setbox\z@\hbox{%
2920       \expandafter\char\csname\fontencoding dqpos\endcsname}%
2921       \dimen@ -.45ex\advance\dimen@\ht\z@
2922       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2923     \expandafter\accent\csname\fontencoding dqpos\endcsname
2924     \fontdimen5\font\U@D #1%
2925   \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).

```
2926 \AtBeginDocument{%
2927   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2928   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2929   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2930   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{i}}%
2931   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
2932   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
2933   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
```

```

2934 \DeclareTextCompositeCommand{"}{OT1}{E}{\bbl@umlaut{E}}%
2935 \DeclareTextCompositeCommand{"}{OT1}{I}{\bbl@umlaut{I}}%
2936 \DeclareTextCompositeCommand{"}{OT1}{O}{\bbl@umlaut{O}}%
2937 \DeclareTextCompositeCommand{"}{OT1}{U}{\bbl@umlaut{U}}

```

Finally, make sure the default hyphenrules are defined (even if empty). For internal use, another empty `\language` is defined. Currently used in Amharic.

```

2938 \ifx\l@english\@undefined
2939 \chardef\l@english\z@
2940 \fi
2941 % The following is used to cancel rules in ini files (see Amharic).
2942 \ifx\l@babelnohyphens\@undefined
2943 \newlanguage\l@babelnohyphens
2944 \fi

```

## 9.13 Layout

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

```

2945 \bbl@trace{Bidi layout}
2946 \providecommand\IfBabelLayout[3]{#3}%
2947 \newcommand\BabelPatchSection[1]{%
2948   \@ifundefined{#1}{}{%
2949     \bbl@exp{\let\bbl@ss@#1<\<#1>%
2950       \@namedef{#1}{%
2951         \@ifstar{\bbl@presec@#1}{%
2952           {\@dblarg{\bbl@presec@x{#1}}}}}%
2953 \def\bbl@presec@x#1[#2]#3{%
2954   \bbl@exp{%
2955     \\select@language@x{\bbl@main@language}%
2956     \\bbl@cs{sspre@#1}%
2957     \\bbl@cs{ss@#1}%
2958     [\\foreignlanguage{\language}{\unexpanded{#2}}}%
2959     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2960     \\select@language@x{\language}}}
2961 \def\bbl@presec@#1#2{%
2962   \bbl@exp{%
2963     \\select@language@x{\bbl@main@language}%
2964     \\bbl@cs{sspre@#1}%
2965     \\bbl@cs{ss@#1}*%
2966     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
2967     \\select@language@x{\language}}}
2968 \IfBabelLayout{sectioning}%
2969   {\BabelPatchSection{part}%
2970    \BabelPatchSection{chapter}%
2971    \BabelPatchSection{section}%
2972    \BabelPatchSection{subsection}%
2973    \BabelPatchSection{subsubsection}%
2974    \BabelPatchSection{paragraph}%
2975    \BabelPatchSection{subparagraph}%
2976    \def\babel@toc#1{%
2977      \select@language@x{\bbl@main@language}}}%
2978 \IfBabelLayout{captions}%
2979   {\BabelPatchSection{caption}}}%

```

## 9.14 Load engine specific macros

```

2980 \bbl@trace{Input engine specific macros}
2981 \ifcase\bbl@engine

```

```

2982 \input txtbabel.def
2983 \or
2984 \input luababel.def
2985 \or
2986 \input xebabel.def
2987 \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.

```

2988 \bbl@trace{Creating languages and reading ini files}
2989 \newcommand\babelprovide[2][{}%
2990   \let\bbl@savelangname\language
2991   \edef\bbl@savelocaleid{\the\localeid}%
2992   % Set name and locale id
2993   \edef\language{#2}%
2994   % \global\@namedef\bbl@lcname@#2}{#2}%
2995   \bbl@id@assign
2996   \let\bbl@KVP@captions\@nil
2997   \let\bbl@KVP@date\@nil
2998   \let\bbl@KVP@import\@nil
2999   \let\bbl@KVP@main\@nil
3000   \let\bbl@KVP@script\@nil
3001   \let\bbl@KVP@language\@nil
3002   \let\bbl@KVP@hyphenrules\@nil
3003   \let\bbl@KVP@mapfont\@nil
3004   \let\bbl@KVP@maparabic\@nil
3005   \let\bbl@KVP@mapdigits\@nil
3006   \let\bbl@KVP@intraspace\@nil
3007   \let\bbl@KVP@intrapenalty\@nil
3008   \let\bbl@KVP@onchar\@nil
3009   \let\bbl@KVP@alph\@nil
3010   \let\bbl@KVP@Alph\@nil
3011   \let\bbl@KVP@labels\@nil
3012   \bbl@csarg\let{KVP@labels*}\@nil
3013   \bbl@forkv{#1}{% TODO - error handling
3014     \in{/{/}{##1}%
3015     \ifin@
3016       \bbl@renewinikey##1\@{##2}%
3017     \else
3018       \bbl@csarg\def{KVP@##1}{##2}%
3019     \fi}%
3020   % == init ==
3021   \ifx\bbl@screset\@undefined
3022     \bbl@ldfinit
3023   \fi
3024   % ==
3025   \let\bbl@lbkflag\relax % \@empty = do setup linebreak
3026   \bbl@ifunset{date#2}%
3027     {\let\bbl@lbkflag\@empty}% new
3028     {\ifx\bbl@KVP@hyphenrules\@nil\else
3029       \let\bbl@lbkflag\@empty
3030     \fi
3031     \ifx\bbl@KVP@import\@nil\else
3032       \let\bbl@lbkflag\@empty
3033     \fi}%
3034   % == import, captions ==

```

```

3035 \ifx\bb1@KVP@import\@nil\else
3036 \bb1@exp{\bb1@ifblank{\bb1@KVP@import}}%
3037 {\ifx\bb1@initload\relax
3038 \begingroup
3039 \def\BabelBeforeIni##1##2{\gdef\bb1@KVP@import{##1}\endinput}%
3040 \bb1@input@texini{#2}%
3041 \endgroup
3042 \else
3043 \xdef\bb1@KVP@import{\bb1@initload}%
3044 \fi}%
3045 {}%
3046 \fi
3047 \ifx\bb1@KVP@captions\@nil
3048 \let\bb1@KVP@captions\bb1@KVP@import
3049 \fi
3050 % Load ini
3051 \bb1@ifunset{date#2}%
3052 {\bb1@provide@new{#2}}%
3053 {\bb1@ifblank{#1}%
3054 {}% With \bb1@load@basic below
3055 {\bb1@provide@renew{#2}}}%
3056 % Post tasks
3057 % -----
3058 % == ensure captions ==
3059 \ifx\bb1@KVP@captions\@nil\else
3060 \bb1@ifunset{\bb1@extracaps@#2}%
3061 {\bb1@exp{\bb1@babelensure[exclude=\\today]{#2}}}%
3062 {\toks@\expandafter\expandafter\expandafter
3063 {\csname \bb1@extracaps@#2\endcsname}%
3064 \bb1@exp{\bb1@babelensure[exclude=\\today,include=\the\toks@]{#2}}%
3065 \bb1@ifunset{\bb1@ensure@\language}%
3066 {\bb1@exp{%
3067 \\\DeclareRobustCommand\<\bb1@ensure@\language>[1]{%
3068 \\\foreignlanguage{\language}%
3069 {####1}}}%
3070 {}%
3071 \bb1@exp{%
3072 \\\bb1@tglobal\<\bb1@ensure@\language>%
3073 \\\bb1@tglobal\<\bb1@ensure@\language\space>}%
3074 \fi
3075 % ==
3076 % At this point all parameters are defined if 'import'. Now we
3077 % execute some code depending on them. But what about if nothing was
3078 % imported? We just set the very basic parameters.
3079 \bb1@load@basic{#2}%
3080 % == script, language ==
3081 % Override the values from ini or defines them
3082 \ifx\bb1@KVP@script\@nil\else
3083 \bb1@csarg\edef{sname#2}{\bb1@KVP@script}%
3084 \fi
3085 \ifx\bb1@KVP@language\@nil\else
3086 \bb1@csarg\edef{lname#2}{\bb1@KVP@language}%
3087 \fi
3088 % == onchar ==
3089 \ifx\bb1@KVP@onchar\@nil\else
3090 \bb1@luahyphenate
3091 \directlua{
3092 if Babel.locale_mapped == nil then
3093 Babel.locale_mapped = true

```



```

3094     Babel.linebreaking.add_before(Babel.locale_map)
3095     Babel.loc_to_scr = {}
3096     Babel.chr_to_loc = Babel.chr_to_loc or {}
3097   end}%
3098   \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
3099   \ifin@
3100     \ifx\bbl@starthyphens\undefined % Needed if no explicit selection
3101       \AddBabelHook{babel-onchar}{beforestart}{{\bbl@starthyphens}}%
3102     \fi
3103     \bbl@exp{\bbl@add\bbl@starthyphens
3104       {\bbl@patterns@lua{\language}}}%
3105     % TODO - error/warning if no script
3106     \directlua{
3107       if Babel.script_blocks['\bbl@cl{sbc}'] then
3108         Babel.loc_to_scr[\the\localeid] =
3109           Babel.script_blocks['\bbl@cl{sbc}']
3110         Babel.locale_props[\the\localeid].lc = \the\localeid\space
3111         Babel.locale_props[\the\localeid].lg = \the\nameuse{1@\language}\space
3112       end
3113     }%
3114   \fi
3115   \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
3116   \ifin@
3117     \bbl@ifunset{bbl@lsys@\language}{\bbl@provide@lsys{\language}}}%
3118     \bbl@ifunset{bbl@wdir@\language}{\bbl@provide@dirs{\language}}}%
3119     \directlua{
3120       if Babel.script_blocks['\bbl@cl{sbc}'] then
3121         Babel.loc_to_scr[\the\localeid] =
3122           Babel.script_blocks['\bbl@cl{sbc}']
3123       end}%
3124     \ifx\bbl@mapselect\undefined
3125       \AtBeginDocument{%
3126         \expandafter\bbl@add\csname selectfont \endcsname{{\bbl@mapselect}}%
3127         {\selectfont}}%
3128       \def\bbl@mapselect{%
3129         \let\bbl@mapselect\relax
3130         \edef\bbl@prefontid{\fontid\font}}%
3131       \def\bbl@mapdir##1{%
3132         {\def\language{##1}%
3133          \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
3134          \bbl@switchfont
3135          \directlua{
3136            Babel.locale_props[\the\csname bbl@id@##1\endcsname]
3137              [\bbl@prefontid] = \fontid\font\space}}}%
3138       \fi
3139       \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
3140     \fi
3141     % TODO - catch non-valid values
3142   \fi
3143   % == mapfont ==
3144   % For bidi texts, to switch the font based on direction
3145   \ifx\bbl@KVP@mapfont\@nil\else
3146     \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}}%
3147     {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\
3148       mapfont. Use 'direction'.%
3149       {See the manual for details.}}}%
3150     \bbl@ifunset{bbl@lsys@\language}{\bbl@provide@lsys{\language}}}%
3151     \bbl@ifunset{bbl@wdir@\language}{\bbl@provide@dirs{\language}}}%
3152     \ifx\bbl@mapselect\undefined

```

```

3153 \AtBeginDocument{%
3154   \expandafter\bbbl@add\csname selectfont \endcsname{\bbbl@mapselect}}%
3155   {\selectfont}}%
3156 \def\bbbl@mapselect{%
3157   \let\bbbl@mapselect\relax
3158   \edef\bbbl@prefontid{\fontid\font}}%
3159 \def\bbbl@mapdir##1{%
3160   {\def\language{##1}%
3161    \let\bbbl@ifrestoring\@firstoftwo % avoid font warning
3162    \bbbl@switchfont
3163    \directlua{Babel.fontmap
3164      [\the\csname bbl@wdir@##1\endcsname]%
3165      [\bbbl@prefontid]=\fontid\font}}}%
3166 \fi
3167 \bbbl@exp{\bbbl@add\bbbl@mapselect{\bbbl@mapdir{\language}}}%
3168 \fi
3169 % == Line breaking: intraspace, intrapenalty ==
3170 % For CJK, East Asian, Southeast Asian, if interspace in ini
3171 \ifx\bbbl@KVP@intraspace\@nil\else % We can override the ini or set
3172   \bbbl@csarg\edef{intsp#2}{\bbbl@KVP@intraspace}%
3173 \fi
3174 \bbbl@provide@intraspace
3175 % == Line breaking: hyphenate.other.locale/.script==
3176 \ifx\bbbl@lbkflag\@empty
3177   \bbbl@ifunset{\bbbl@hyotl@language}{}%
3178   {\bbbl@csarg\bbbl@replace{\hyotl@language}{ }{,}}%
3179   \bbbl@startcommands*{\language}{}%
3180   \bbbl@csarg\bbbl@foreach{\hyotl@language}{%
3181     \ifcase\bbbl@engine
3182       \ifnum##1<257
3183         \SetHyphenMap{\BabelLower{##1}{##1}}%
3184       \fi
3185     \else
3186       \SetHyphenMap{\BabelLower{##1}{##1}}%
3187     \fi}%
3188   \bbbl@endcommands}%
3189 \bbbl@ifunset{\bbbl@hyots@language}{}%
3190 {\bbbl@csarg\bbbl@replace{\hyots@language}{ }{,}}%
3191 \bbbl@csarg\bbbl@foreach{\hyots@language}{%
3192   \ifcase\bbbl@engine
3193     \ifnum##1<257
3194       \global\lccode##1=##1\relax
3195     \fi
3196   \else
3197     \global\lccode##1=##1\relax
3198   \fi}}%
3199 \fi
3200 % == Counters: maparabic ==
3201 % Native digits, if provided in ini (TeX level, xe and lua)
3202 \ifcase\bbbl@engine\else
3203   \bbbl@ifunset{\bbbl@dgnat@language}{}%
3204   {\expandafter\ifx\csname bbl@dgnat@language\endcsname\@empty\else
3205     \expandafter\expandafter\expandafter
3206     \bbbl@setdigits\csname bbl@dgnat@language\endcsname
3207     \ifx\bbbl@KVP@maparabic\@nil\else
3208       \ifx\bbbl@latinarabic\@undefined
3209         \expandafter\let\expandafter\@arabic
3210         \csname bbl@counter@language\endcsname
3211       \else % ie, if layout=counters, which redefines \@arabic

```

```

3212         \expandafter\let\expandafter\bbl@latinarabic
3213         \csname bbl@counter@\language\endcsname
3214     \fi
3215     \fi
3216 \fi}%
3217 \fi
3218 % == Counters: mapdigits ==
3219 % Native digits (lua level).
3220 \ifodd\bbl@engine
3221     \ifx\bbl@KVP@mapdigits@nil\else
3222         \bbl@ifunset{\bbl@dgnat@\language\endcsname}{}%
3223         {\RequirePackage{luatexbase}%
3224         \bbl@activate@preotf
3225         \directlua{
3226             Babel = Babel or {} %% -> presets in luababel
3227             Babel.digits_mapped = true
3228             Babel.digits = Babel.digits or {}
3229             Babel.digits[\the\localeid] =
3230                 table.pack(string.utfvalue('\bbl@cl{dgnat}'))
3231             if not Babel.numbers then
3232                 function Babel.numbers(head)
3233                     local LOCALE = luatexbase.registernumber'bbl@attr@locale'
3234                     local GLYPH = node.id'glyph'
3235                     local inmath = false
3236                     for item in node.traverse(head) do
3237                         if not inmath and item.id == GLYPH then
3238                             local temp = node.get_attribute(item, LOCALE)
3239                             if Babel.digits[temp] then
3240                                 local chr = item.char
3241                                 if chr > 47 and chr < 58 then
3242                                     item.char = Babel.digits[temp][chr-47]
3243                                 end
3244                             end
3245                             elseif item.id == node.id'math' then
3246                                 inmath = (item.subtype == 0)
3247                             end
3248                         end
3249                     return head
3250                 end
3251             end
3252         }}%
3253     \fi
3254 \fi
3255 % == Counters: alph, Alph ==
3256 % What if extras<lang> contains a \babel@save\@alph? It won't be
3257 % restored correctly when exiting the language, so we ignore
3258 % this change with the \bbl@alph@saved trick.
3259 \ifx\bbl@KVP@alph@nil\else
3260     \toks@\expandafter\expandafter\expandafter{%
3261         \csname extras\language\endcsname}%
3262     \bbl@exp{%
3263         \def\<extras\language>{%
3264             \let\\\bbl@alph@saved\\\@alph
3265             \the\toks@
3266             \let\\\@alph\\\bbl@alph@saved
3267             \\\babel@save\\\@alph
3268             \let\\\@alph\<bbl@cntr@\bbl@KVP@alph @\language>}}%
3269 \fi
3270 \ifx\bbl@KVP@Alph@nil\else

```

```

3271 \toks@\expandafter\expandafter\expandafter{%
3272 \csname extras\language\endcsname}%
3273 \bbl@exp{%
3274 \def\<extras\language>{%
3275 \let\\bbl@Alph@saved\\@Alph
3276 \the\toks@
3277 \let\\@Alph\\bbl@Alph@saved
3278 \\babel@save\\@Alph
3279 \let\\@Alph\<bbl@cntr@bbl@KVP@Alph @\language>}}%
3280 \fi
3281 % == require.babel in ini ==
3282 % To load or reload the babel-*.tex, if require.babel in ini
3283 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
3284 \bbl@ifunset{bbl@rqtex@\language}{}%
3285 {\expandafter\ifx\csname bbl@rqtex@\language\endcsname\@empty\else
3286 \let\BabelBeforeIni@gobbletwo
3287 \chardef\atcatcode=\catcode`\@
3288 \catcode`\@=11\relax
3289 \bbl@input@texini{\bbl@cs{rqtex@\language}}%
3290 \catcode`\@=\atcatcode
3291 \let\atcatcode\relax
3292 \fi}%
3293 \fi
3294 % == main ==
3295 \ifx\bbl@KVP@main\@nil % Restore only if not 'main'
3296 \let\language\bbl@savelangname
3297 \chardef\localeid\bbl@savelocaleid\relax
3298 \fi}

```

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

```

3299 \def\bbl@provide@new#1{%
3300 \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
3301 \@namedef{extras#1}{}%
3302 \@namedef{noextras#1}{}%
3303 \bbl@startcommands*{#1}{captions}%
3304 \ifx\bbl@KVP@captions\@nil % and also if import, implicit
3305 \def\bbl@tempb##1% elt for \bbl@captionslist
3306 \ifx##1\@empty\else
3307 \bbl@exp{%
3308 \\SetString\\##1{%
3309 \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
3310 \expandafter\bbl@tempb
3311 \fi}%
3312 \expandafter\bbl@tempb\bbl@captionslist\@empty
3313 \else
3314 \ifx\bbl@initoload\relax
3315 \bbl@read@ini{\bbl@KVP@captions}0% Here letters cat = 11
3316 \else
3317 \bbl@read@ini{\bbl@initoload}0% Here all letters cat = 11
3318 \fi
3319 \bbl@after@ini
3320 \bbl@savestrings
3321 \fi
3322 \StartBabelCommands*{#1}{date}%
3323 \ifx\bbl@KVP@import\@nil
3324 \bbl@exp{%
3325 \\SetString\\today{\\bbl@nocaption{today}{#1today}}}%
3326 \else
3327 \bbl@savetoday

```

```

3328 \bbl@savedate
3329 \fi
3330 \bbl@endcommands
3331 \bbl@load@basic{#1}%
3332 % == hyphenmins == (only if new)
3333 \bbl@exp{%
3334 \gdef\<#1hyphenmins>{%
3335 {\bbl@ifunset{\bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
3336 {\bbl@ifunset{\bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}%
3337 % == hyphenrules ==
3338 \bbl@provide@hyphens{#1}%
3339 % == frenchspacing == (only if new)
3340 \bbl@ifunset{\bbl@frspc@#1}{}%
3341 {\edef\bbl@tempa{\bbl@cl{frspc}}}%
3342 \edef\bbl@tempa{\expandafter\car\bbl@tempa\@nil}%
3343 \if u\bbl@tempa % do nothing
3344 \else\if n\bbl@tempa % non french
3345 \expandafter\bbl@add\csname extras#1\endcsname{%
3346 \let\bbl@elt\bbl@fs@elt@i
3347 \bbl@fs@chars}%
3348 \else\if y\bbl@tempa % french
3349 \expandafter\bbl@add\csname extras#1\endcsname{%
3350 \let\bbl@elt\bbl@fs@elt@ii
3351 \bbl@fs@chars}%
3352 \fi\fi\fi}%
3353 %
3354 \ifx\bbl@KVP@main\@nil\else
3355 \expandafter\main@language\expandafter{#1}%
3356 \fi}
3357 % A couple of macros used above, to avoid hashes #####...
3358 \def\bbl@fs@elt@i#1#2#3{%
3359 \ifnum\sfcode`#1=#2\relax
3360 \babel@savevariable{\sfcode`#1}%
3361 \sfcode`#1=#3\relax
3362 \fi}%
3363 \def\bbl@fs@elt@ii#1#2#3{%
3364 \ifnum\sfcode`#1=#3\relax
3365 \babel@savevariable{\sfcode`#1}%
3366 \sfcode`#1=#2\relax
3367 \fi}%
3368 %
3369 \def\bbl@provide@renew#1{%
3370 \ifx\bbl@KVP@captions\@nil\else
3371 \StartBabelCommands*{#1}{captions}%
3372 \bbl@read@ini{\bbl@KVP@captions}0% Here all letters cat = 11
3373 \bbl@after@ini
3374 \bbl@savestrings
3375 \EndBabelCommands
3376 \fi
3377 \ifx\bbl@KVP@import\@nil\else
3378 \StartBabelCommands*{#1}{date}%
3379 \bbl@savetoday
3380 \bbl@savedate
3381 \EndBabelCommands
3382 \fi
3383 % == hyphenrules ==
3384 \ifx\bbl@lbkflag\@empty
3385 \bbl@provide@hyphens{#1}%
3386 \fi}

```

```

3387% Load the basic parameters (ids, typography, counters, and a few
3388% more), while captions and dates are left out. But it may happen some
3389% data has been loaded before automatically, so we first discard the
3390% saved values.
3391 \def\bbbl@linebreak@export{%
3392   \bbbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3393   \bbbl@exportkey{hyphr}{typography.hyphenrules}{}%
3394   \bbbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
3395   \bbbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3396   \bbbl@exportkey{prehc}{typography.prehyphenchar}{}%
3397   \bbbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3398   \bbbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3399   \bbbl@exportkey{intsp}{typography.intraspaces}{}%
3400   \bbbl@exportkey{chrng}{characters.ranges}{}%
3401 \def\bbbl@load@basic#1{%
3402   \bbbl@ifunset{bbbl@inidata@\language\name}{}%
3403   {\getlocaleproperty\bbbl@tempa{\language\name}{identification/load.level}%
3404     \ifcase\bbbl@tempa\else
3405       \bbbl@csarg\let{lname@\language\name}\relax
3406     \fi}%
3407   \bbbl@ifunset{bbbl@lname@#1}%
3408   {\def\BabelBeforeIni##1##2{%
3409     \begingroup
3410       \let\bbbl@ini@captions@aux\@gobbletwo
3411       \def\bbbl@inidate ####1.####2.####3.####4\relax ####5####6{%
3412         \bbbl@read@ini{##1}0%
3413         \bbbl@linebreak@export
3414         \bbbl@exportkey{dgnat}{numbers.digits.native}{}%
3415         \bbbl@exportkey{frspc}{typography.frenchspacing}{u}% unset
3416         \ifx\bbbl@initoload\relax\endinput\fi
3417       \endgroup}%
3418     \begingroup % boxed, to avoid extra spaces:
3419       \ifx\bbbl@initoload\relax
3420         \bbbl@input@texini{##1}%
3421       \else
3422         \setbox\z@\hbox{\BabelBeforeIni{\bbbl@initoload}}}%
3423       \fi
3424     \endgroup}%
3425   {}}

```

The hyphenrules option is handled with an auxiliary macro.

```

3426 \def\bbbl@provide@hyphens#1{%
3427   \let\bbbl@tempa\relax
3428   \ifx\bbbl@KVP@hyphenrules\@nil\else
3429     \bbbl@replace\bbbl@KVP@hyphenrules{ }{,}%
3430     \bbbl@foreach\bbbl@KVP@hyphenrules{%
3431       \ifx\bbbl@tempa\relax % if not yet found
3432         \bbbl@ifsamestring{##1}{+}%
3433         {\bbbl@exp{\addlanguage\<l@##1>}}}%
3434       {}%
3435       \bbbl@ifunset{l@##1}%
3436       {}%
3437       {\bbbl@exp{\let\bbbl@tempa\<l@##1>}}}%
3438     \fi}%
3439 \fi
3440 \ifx\bbbl@tempa\relax % if no opt or no language in opt found
3441   \ifx\bbbl@KVP@import\@nil
3442     \ifx\bbbl@initoload\relax\else
3443       \bbbl@exp{ % and hyphenrules is not empty

```

```

3444      \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
3445      {}%
3446      {\let\\bbl@tempa<l@bbl@cl{hyphr}>}}%
3447  \fi
3448  \else % if importing
3449      \bbl@exp{%
3450          \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
3451          {}%
3452          {\let\\bbl@tempa<l@bbl@cl{hyphr}>}}%
3453  \fi
3454  \fi
3455  \bbl@ifunset{bbl@tempa}%
3456      { \bbl@ifunset{l@#1}%
3457          { \bbl@exp{\\adddialect<l@#1>\language}}%
3458          {}%
3459          { \bbl@exp{\\adddialect<l@#1>\bbl@tempa}}%
3460      }%
3461      ie, relax or undefined
3462      no hyphenrules found - fallback
3463      so, l@<lang> is ok - nothing to do
3464      found in opt list or ini
3465
3466  The reader of ini files. There are 3 possible cases: a section name (in the form [ . . . ]), a comment
3467  (starting with ;) and a key/value pair.
3468
3469  \ifx\bbl@readstream@undefined
3470  \csname newread\endcsname\bbl@readstream
3471  \fi
3472  \def\bbl@input@texini#1{%
3473      \bbl@bsphack
3474      \bbl@exp{%
3475          \catcode`\\=14 \catcode`\\=0
3476          \catcode`\\{=1 \catcode`\\}=2
3477          \lowercase{\\InputIfFileExists{babel-#1.tex}{}}%
3478          \catcode`\\=\\the\catcode`%\relax
3479          \catcode`\\=\\the\catcode`%\relax
3480          \catcode`\\{=\\the\catcode`%\relax
3481          \catcode`\\}=\\the\catcode`%\relax}%
3482      \bbl@esphack}
3483  \def\bbl@inipreread#1=#2\@@{%
3484      \bbl@trim@def\bbl@tempa{#1}% Redundant below !!
3485      \bbl@trim\toks@{#2}%
3486      % Move trims here ??
3487      \bbl@ifunset{bbl@KVP@bbl@section/\bbl@tempa}%
3488      { \bbl@exp{%
3489          \\g@addto@macro\\bbl@inidata{%
3490              \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
3491          \expandafter\bbl@inireader\bbl@tempa=#2\@@}%
3492      }%
3493  \def\bbl@fetch@ini#1#2{%
3494      \bbl@exp{\def\\bbl@inidata{%
3495          \\bbl@elt{identification}{tag.ini}{#1}%
3496          \\bbl@elt{identification}{load.level}{#2}}}%
3497      \openin\bbl@readstream=babel-#1.ini
3498      \ifeof\bbl@readstream
3499          \bbl@error
3500          {There is no ini file for the requested language\\
3501            (#1). Perhaps you misspelled it or your installation\\
3502            is not complete.}%
3503          {Fix the name or reinstall babel.}%
3504      \else
3505          \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
3506          \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
3507          \bbl@info{Importing
3508              \ifcase#2 \or font and identification \or basic \fi

```

```

3500             data for \language\%
3501             from babel-#1.ini. Reported}%
3502     \loop
3503     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3504     \endlinechar\m@ne
3505     \read\bbl@readstream to \bbl@line
3506     \endlinechar`\^^M
3507     \ifx\bbl@line\@empty\else
3508     \expandafter\bbl@inline\bbl@line\bbl@inline
3509     \fi
3510     \repeat
3511 \fi}
3512 \def\bbl@read@ini#1#2{%
3513 \bbl@csarg\xdef{lini@\language}\{#1}%
3514 \let\bbl@section\@empty
3515 \let\bbl@savestrings\@empty
3516 \let\bbl@savetoday\@empty
3517 \let\bbl@savedate\@empty
3518 \let\bbl@inireader\bbl@iniskip
3519 \bbl@fetch@ini{#1}{#2}%
3520 \bbl@foreach\bbl@renewlist{%
3521 \bbl@ifunset{\bbl@renew@##1}{\bbl@inisec[##1]\@{}}%
3522 \global\let\bbl@renewlist\@empty
3523 % Ends last section. See \bbl@inisec
3524 \def\bbl@elt##1##2{\bbl@inireader##1=##2\@{}}%
3525 \bbl@cs{renew@\bbl@section}%
3526 \global\bbl@csarg\let{renew@\bbl@section}\relax
3527 \bbl@cs{secpost@\bbl@section}%
3528 \bbl@csarg{\global\expandafter\let}{inidata@\language}\bbl@inidata
3529 \bbl@exp{\bbl@add@list\bbl@ini@loaded{\language}}%
3530 \bbl@to\global\bbl@ini@loaded}
3531 \def\bbl@inline#1\bbl@inline{%
3532 \@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. By default, key=val pairs are ignored. The secpost “hook” is used only by ‘identification’, while secpre only by date.gregorian.licr.

```

3533 \def\bbl@iniskip#1\@{}}%      if starts with ;
3534 \def\bbl@inisec[##1]#2\@{}}%  if starts with opening bracket
3535 \def\bbl@elt##1##2{%
3536 \expandafter\toks@\expandafter{%
3537 \expandafter{\bbl@section}{##1}{##2}}%
3538 \bbl@exp{%
3539 \g@addto@macro\bbl@inidata{\bbl@elt\the\toks@}}%
3540 \bbl@inireader##1=##2\@{}}%
3541 \bbl@cs{renew@\bbl@section}%
3542 \global\bbl@csarg\let{renew@\bbl@section}\relax
3543 \bbl@cs{secpost@\bbl@section}%
3544 % The previous code belongs to the previous section.
3545 % -----
3546 % Now start the current one.
3547 \in@{=date.}\{#1}%
3548 \ifin@
3549 \lowercase{\def\bbl@tempa{=#1}}%
3550 \bbl@replace\bbl@tempa{=date.gregorian}\{}%
3551 \bbl@replace\bbl@tempa{=date.}\{}%
3552 \in@{.licr}\{#1}%
3553 \ifin@

```



```

3554 \ifcase\bb1@engine
3555 \bb1@replace\bb1@tempa{.licr=}{}%
3556 \else
3557 \let\bb1@tempa\relax
3558 \fi
3559 \fi
3560 \ifx\bb1@tempa\relax\else
3561 \bb1@replace\bb1@tempa{=}{}%
3562 \bb1@exp{%
3563 \def\<bb1@inikv@#1>####1=####2\\@{%
3564 \\bb1@inidate####1...\relax{####2}{\bb1@tempa}}}%
3565 \fi
3566 \fi
3567 \def\bb1@section{#1}%
3568 \def\bb1@elt##1##2{%
3569 \@namedef{bb1@KVP@#1/#1}{}}%
3570 \bb1@cs{renew@#1}%
3571 \bb1@cs{secp@#1}% pre-section `hook'
3572 \bb1@ifunset{bb1@inikv@#1}%
3573 {\let\bb1@inireader\bb1@iniskip}%
3574 {\bb1@exp{\let\\bb1@inireader\<bb1@inikv@#1>}}
3575 \let\bb1@renewlist\@empty
3576 \def\bb1@renewinikv#1/#2\\@#3{%
3577 \bb1@ifunset{bb1@renew@#1}%
3578 {\bb1@add@list\bb1@renewlist{#1}}%
3579 {}}%
3580 \bb1@csarg\bb1@add{renew@#1}{\bb1@elt{#2}{#3}}

```

Reads a key=val line and stores the trimmed val in \bb1@kv@<section>.<key>.

```

3581 \def\bb1@inikv#1=#2\\@{% key=value
3582 \bb1@trim@def\bb1@tempa{#1}%
3583 \bb1@trim\toks@{#2}%
3584 \bb1@csarg\edef{kv@\bb1@section.\bb1@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.

```

3585 \def\bb1@exportkey#1#2#3{%
3586 \bb1@ifunset{bb1@kv@#2}%
3587 {\bb1@csarg\gdef{#1@\language}\{#3}}%
3588 {\expandafter\ifx\csname bb1@kv@#2\endcsname\@empty
3589 \bb1@csarg\gdef{#1@\language}\{#3}}%
3590 \else
3591 \bb1@exp{\global\let\<bb1@#1@\language>\<bb1@kv@#2>}}%
3592 \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 \bb1@secp@identification is called always (via \bb1@inisec), while \bb1@after@ini must be called explicitly after \bb1@read@ini if necessary.

```

3593 \def\bb1@iniwarning#1{%
3594 \bb1@ifunset{bb1@kv@identification.warning#1}{}%
3595 {\bb1@warning{%
3596 From babel-\bb1@cs{lini@\language}.ini:\\
3597 \bb1@cs{kv@identification.warning#1}\\
3598 Reported }}}
3599 %
3600 \let\bb1@inikv@identification\bb1@inikv
3601 \def\bb1@secp@identification{%
3602 \bb1@iniwarning}%

```

```

3603 \ifcase\bb1@engine
3604   \bb1@iniwarning{.pdflatex}%
3605 \or
3606   \bb1@iniwarning{.lualatex}%
3607 \or
3608   \bb1@iniwarning{.xelatex}%
3609 \fi%
3610 \bb1@exportkey{elname}{identification.name.english}{}%
3611 \bb1@exp{\bb1@exportkey{lname}{identification.name.opentype}%
3612   {\csname bbl@elname\language\endcsname}}%
3613 \bb1@exportkey{tbcpl}{identification.tag.bcp47}{}%
3614 \bb1@exportkey{lbcpl}{identification.language.tag.bcp47}{}%
3615 \bb1@exportkey{lotf}{identification.tag.opentype}{dflt}%
3616 \bb1@exportkey{esname}{identification.script.name}{}%
3617 \bb1@exp{\bb1@exportkey{sname}{identification.script.name.opentype}%
3618   {\csname bbl@esname\language\endcsname}}%
3619 \bb1@exportkey{sbcpl}{identification.script.tag.bcp47}{}%
3620 \bb1@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3621 \ifbbl@bcptoname
3622   \bb1@csarg\xdef{bcp@map@bbl@cl{tbcpl}}{\language}%
3623 \fi}

```

By default, the following sections are just read. Actions are taken later.

```

3624 \let\bb1@inikv@typography\bb1@inikv
3625 \let\bb1@inikv@characters\bb1@inikv
3626 \let\bb1@inikv@numbers\bb1@inikv

```

Additive numerals require an additional definition. When .1 is found, two macros are defined – the basic one, without .1 called by \localenumeral, and another one preserving the trailing .1 for the ‘units’.

```

3627 \def\bb1@inikv@counters#1=#2\@@{%
3628   \bb1@ifsamestring{#1}{digits}%
3629   {\bb1@error{The counter name 'digits' is reserved for mapping\\
3630     decimal digits}%
3631     {Use another name.}}%
3632   }%
3633 \def\bb1@tempc{#1}%
3634 \bb1@trim@def{\bb1@tempb*}{#2}%
3635 \in@{.1$}{#1$}%
3636 \ifin@
3637   \bb1@replace\bb1@tempc{.1}{}%
3638   \bb1@csarg\protected@xdef{cntr@bbl@tempc @\language}{%
3639     \noexpand\bb1@alphanumeric{\bb1@tempc}}%
3640 \fi
3641 \in@{.F.}{#1}%
3642 \ifin@else\in@{.S.}{#1}\fi
3643 \ifin@
3644   \bb1@csarg\protected@xdef{cntr@#1@\language}{\bb1@tempb*}%
3645 \else
3646   \toks@{}% Required by \bb1@buildifcase, which returns \bb1@tempa
3647   \expandafter\bb1@buildifcase\bb1@tempb* \ \ % Space after \
3648   \bb1@csarg{\global\expandafter\let}{cntr@#1@\language}\bb1@tempa
3649 \fi}
3650 \def\bb1@after@ini{%
3651   \bb1@linebreak@export
3652   \bb1@exportkey{dgnat}{numbers.digits.native}{}%
3653   \bb1@exportkey{rqtex}{identification.require.babel}{}%
3654   \bb1@exportkey{frspc}{typography.frenchspacing}{u}% unset
3655   \bb1@toggle\bb1@savetoday

```

```
3656 \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.

```
3657 \ifcase\bbl@engine
3658 \bbl@csarg\def{inikv@captions.licr}#1=#2\@@{%
3659 \bbl@ini@captions@aux{#1}{#2}}
3660 \else
3661 \def\bbl@inikv@captions#1=#2\@@{%
3662 \bbl@ini@captions@aux{#1}{#2}}
3663 \fi
```

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

```
3664 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3665 \bbl@replace\bbl@tempa{.template}{}%
3666 \def\bbl@toreplace{#1}{}%
3667 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
3668 \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3669 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3670 \bbl@replace\bbl@toreplace{[ ]}{name\endcsname}}%
3671 \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
3672 \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3673 \ifin@
3674 \@nameuse{bbl@patch\bbl@tempa}%
3675 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3676 \fi
3677 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3678 \ifin@
3679 \toks@\expandafter{\bbl@toreplace}%
3680 \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}%
3681 \fi}
3682 \def\bbl@ini@captions@aux#1#2{%
3683 \bbl@trim@def\bbl@tempa{#1}%
3684 \bbl@xin@{.template}{\bbl@tempa}%
3685 \ifin@
3686 \bbl@ini@captions@template{#2}\language name
3687 \else
3688 \bbl@ifblank{#2}%
3689 {\bbl@exp{%
3690 \toks@{\bbl@nocaption{\bbl@tempa}{\language name\bbl@tempa name}}}%
3691 {\bbl@trim\toks@{#2}}}%
3692 \bbl@exp{%
3693 \bbl@add\bbl@savestrings{%
3694 \SetString\<\bbl@tempa name>{\the\toks@}}}%
3695 \toks@\expandafter{\bbl@captionslist}%
3696 \bbl@exp{\in@{\<\bbl@tempa name>}{\the\toks@}}}%
3697 \ifin@ \else
3698 \bbl@exp{%
3699 \bbl@add\<\bbl@extracaps@\language name>{\<\bbl@tempa name>}}%
3700 \bbl@toglobal\<\bbl@extracaps@\language name>}}%
3701 \fi
3702 \fi}
```

**Labels.** Captions must contain just strings, no format at all, so there is new group in ini files.

```
3703 \def\bbl@list@the{%
3704 part,chapter,section,subsection,subsubsection,paragraph,%
3705 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3706 table,page,footnote,mpfootnote,mpfn}
```

```

3707 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3708 \bbl@ifunset\bbl@map@#1@\language\name}%
3709 {\@nameuse{#1}}%
3710 {\@nameuse\bbl@map@#1@\language\name}}%
3711 \def\bbl@inikv@labels#1=#2\@{%
3712 \in@{.map}{#1}%
3713 \ifin@
3714 \ifx\bbl@KVP@labels\@nil\else
3715 \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3716 \ifin@
3717 \def\bbl@tempc{#1}%
3718 \bbl@replace\bbl@tempc{.map}{}%
3719 \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3720 \bbl@exp{%
3721 \gdef\bbl@map@\bbl@tempc @\language\name>%
3722 {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3723 \bbl@foreach\bbl@list@the{%
3724 \bbl@ifunset{the##1}{}%
3725 {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3726 \bbl@exp{%
3727 \\bbl@sreplace\<the##1>%
3728 {\<\bbl@tempc>{##1}}{\bbl@map@cnt{\bbl@tempc}{##1}}%
3729 \\bbl@sreplace\<the##1>%
3730 {\<\empty @\bbl@tempc>\<c##1>}{\bbl@map@cnt{\bbl@tempc}{##1}}}%
3731 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3732 \toks@\expandafter\expandafter\expandafter{%
3733 \csname the##1\endcsname}%
3734 \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3735 \fi}}%
3736 \fi
3737 \fi
3738 %
3739 \else
3740 %
3741 % The following code is still under study. You can test it and make
3742 % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3743 % language dependent.
3744 \in@{enumerate.}{#1}%
3745 \ifin@
3746 \def\bbl@tempa{#1}%
3747 \bbl@replace\bbl@tempa{enumerate.}{}%
3748 \def\bbl@toreplace{#2}%
3749 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
3750 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3751 \bbl@replace\bbl@toreplace{ ]}{\endcsname}}%
3752 \toks@\expandafter{\bbl@toreplace}%
3753 \bbl@exp{%
3754 \\bbl@add\<extras\language\name>%
3755 \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3756 \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3757 \\bbl@toglobal\<extras\language\name>}%
3758 \fi
3759 \fi}

```

To show correctly some captions in a few languages, we need to patch some internal macros, because the order is hardcoded. For example, in Japanese the chapter number is surrounded by two string, while in Hungarian is placed after. These replacement works in many classes, but not all. Actually, the following lines are somewhat tentative.

```

3760 \def\bbl@chapttype{chapter}

```

```

3761 \ifx\@makechapterhead\undefined
3762 \let\bb1@patchchapter\relax
3763 \else\ifx\thechapter\undefined
3764 \let\bb1@patchchapter\relax
3765 \else\ifx\ps@headings\undefined
3766 \let\bb1@patchchapter\relax
3767 \else
3768 \def\bb1@patchchapter{%
3769 \global\let\bb1@patchchapter\relax
3770 \bb1@add\appendix{\def\bb1@chapttype{appendix}}% Not harmful, I hope
3771 \bb1@toglobal\appendix
3772 \bb1@sreplace\ps@headings
3773 {\@chapapp\thechapter}%
3774 {\bb1@chapterformat}%
3775 \bb1@toglobal\ps@headings
3776 \bb1@sreplace\chaptermark
3777 {\@chapapp\thechapter}%
3778 {\bb1@chapterformat}%
3779 \bb1@toglobal\chaptermark
3780 \bb1@sreplace\@makechapterhead
3781 {\@chapapp\space\thechapter}%
3782 {\bb1@chapterformat}%
3783 \bb1@toglobal\@makechapterhead
3784 \gdef\bb1@chapterformat{%
3785 \bb1@ifunset{bb1@bb1@chapttype fmt@\languagename}%
3786 {\@chapapp\space\thechapter}
3787 {\@nameuse{bb1@bb1@chapttype fmt@\languagename}}}}
3788 \let\bb1@patchappendix\bb1@patchchapter
3789 \fi\fi\fi
3790 \ifx\@part\undefined
3791 \let\bb1@patchpart\relax
3792 \else
3793 \def\bb1@patchpart{%
3794 \global\let\bb1@patchpart\relax
3795 \bb1@sreplace\@part
3796 {\partname\nobreakspace\thepart}%
3797 {\bb1@partformat}%
3798 \bb1@toglobal\@part
3799 \gdef\bb1@partformat{%
3800 \bb1@ifunset{bb1@partfmt@\languagename}%
3801 {\partname\nobreakspace\thepart}
3802 {\@nameuse{bb1@partfmt@\languagename}}}}
3803 \fi

```

**Date.** TODO. Document

```

3804 % Arguments are _not_ protected.
3805 \let\bb1@calendar\@empty
3806 \DeclareRobustCommand\localedate[1][\bb1@localedate{#1}]
3807 \def\bb1@localedate#1#2#3#4{%
3808 \begingroup
3809 \ifx\@empty#1\@empty\else
3810 \let\bb1@ld@calendar\@empty
3811 \let\bb1@ld@variant\@empty
3812 \edef\bb1@tempa{\zap@space#1 \@empty}%
3813 \def\bb1@tempb##1=##2\@{\@namedef{bb1@ld@##1}{##2}}%
3814 \bb1@foreach\bb1@tempa{\bb1@tempb##1\@}%
3815 \edef\bb1@calendar{%
3816 \bb1@ld@calendar
3817 \ifx\bb1@ld@variant\@empty\else

```

```

3818         .\bbl@ld@variant
3819         \fi}%
3820         \bbl@replace\bbl@calendar{gregorian}{}%
3821     \fi
3822     \bbl@cased
3823     {\@nameuse{bbl@date@\languagename @\bbl@calendar}{#2}{#3}{#4}}%
3824 \endgroup}
3825 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3826 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3827     \bbl@trim@def\bbl@tempa{#1.#2}%
3828     \bbl@ifsamestring{\bbl@tempa}{months.wide}%         to savedate
3829     {\bbl@trim@def\bbl@tempa{#3}%
3830         \bbl@trim\toks@{#5}%
3831         \@temptokena\expandafter{\bbl@savestate}%
3832         \bbl@exp{%      Reverse order - in ini last wins
3833             \def\\bbl@savestate{%
3834                 \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3835                 \the\@temptokena}}%
3836         {\bbl@ifsamestring{\bbl@tempa}{date.long}%         defined now
3837             {\lowercase{\def\bbl@tempb{#6}}%
3838                 \bbl@trim@def\bbl@toreplace{#5}%
3839                 \bbl@TG@@date
3840                 \bbl@ifunset{bbl@date@\languagename @}%
3841                 {\global\bbl@csarg\let{date@\languagename @}\bbl@toreplace
3842                     % TODO. Move to a better place.
3843                     \bbl@exp{%
3844                         \gdef\<\languagename date>{\\protect\<\languagename date >}%
3845                         \gdef\<\languagename date >####1####2####3{%
3846                             \\bbl@usedategroupttrue
3847                             \<bbl@ensure@\languagename>{%
3848                                 \\localedate{####1}{####2}{####3}}}%
3849                             \\bbl@add\\bbl@savetoday{%
3850                                 \\SetString\\today{%
3851                                     \<\languagename date>%
3852                                     {\the\year}{\the\month}{\the\day}}}}}%
3853             }%
3854             \ifx\bbl@tempb\@empty\else
3855                 \global\bbl@csarg\let{date@\languagename @\bbl@tempb}\bbl@toreplace
3856             \fi}%
3857         {}}}}

```

**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.

```

3858 \let\bbl@calendar\@empty
3859 \newcommand\BabelDateSpace{\nobreakspace}
3860 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3861 \newcommand\BabelDated[1]{\number#1}
3862 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3863 \newcommand\BabelDateM[1]{\number#1}
3864 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3865 \newcommand\BabelDateMMM[1]{\ifnum#1<10 0\fi\number#1}
3866 \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3867 \newcommand\BabelDatey[1]{\number#1}%
3868 \newcommand\BabelDateyy[1]{\ifnum#1<10 0\number#1 %
3869     \else\ifnum#1<100 \number#1 %
3870     \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3871     \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %

```

```

3873 \else
3874 \bbl@error
3875 {Currently two-digit years are restricted to the\
3876 range 0-9999.}%
3877 {There is little you can do. Sorry.}%
3878 \fi\fi\fi\fi}}
3879 \newcommand\BabelDateyyyy[1]{\number#1}} % FIXME - add leading 0
3880 \def\bbl@replace@finish@iii#1{%
3881 \bbl@exp{\def\#1###1###2###3{\the\toks@}}
3882 \def\bbl@TG@date{%
3883 \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}}%
3884 \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}}%
3885 \bbl@replace\bbl@toreplace{[d]}{\BabelDated{###3}}}%
3886 \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{###3}}}%
3887 \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{###2}}}%
3888 \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{###2}}}%
3889 \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMM{###2}}}%
3890 \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{###1}}}%
3891 \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{###1}}}%
3892 \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{###1}}}%
3893 \bbl@replace\bbl@toreplace{[y|]}{\bbl@datecctr[###1|]}%
3894 \bbl@replace\bbl@toreplace{[m|]}{\bbl@datecctr[###2|]}%
3895 \bbl@replace\bbl@toreplace{[d|]}{\bbl@datecctr[###3|]}%
3896 % Note after \bbl@replace \toks@ contains the resulting string.
3897 % TODO - Using this implicit behavior doesn't seem a good idea.
3898 \bbl@replace@finish@iii\bbl@toreplace}
3899 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3900 \def\bbl@xdatecctr[#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.

```

3901 \def\bbl@provide@lsys#1{%
3902 \bbl@ifunset{bbl@lname@#1}%
3903 {\bbl@ini@basic{#1}}%
3904 {}%
3905 \bbl@csarg\let{lsys@#1}\empty
3906 \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}}%
3907 \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}}%
3908 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}}%
3909 \bbl@ifunset{bbl@lname@#1}{%
3910 {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3911 \ifcase\bbl@engine\or\or
3912 \bbl@ifunset{bbl@prehc@#1}{%
3913 {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3914 {}%
3915 {\ifx\bbl@xenoxyph\undefined
3916 \let\bbl@xenoxyph\bbl@xenoxyph@d
3917 \ifx\AtBeginDocument\notprerr
3918 \expandafter\@secondoftwo % to execute right now
3919 \fi
3920 \AtBeginDocument{%
3921 \expandafter\bbl@add
3922 \csname selectfont \endcsname\bbl@xenoxyph}%
3923 \expandafter\selectlanguage\expandafter{\language}%
3924 \expandafter\bbl@tglobal\csname selectfont \endcsname}%
3925 \fi}}%
3926 \fi
3927 \bbl@csarg\bbl@tglobal{lsys@#1}}
3928 \def\bbl@xenoxyph@d{%

```

```

3929 \bbl@ifset{\bbl@prehc@language}%
3930 {\ifnum\hyphenchar\font=\defaultshyphenchar
3931   \iffontchar\font\bbl@cl{prehc}\relax
3932   \hyphenchar\font\bbl@cl{prehc}\relax
3933   \else\iffontchar\font"200B
3934     \hyphenchar\font"200B
3935   \else
3936     \bbl@warning
3937     {Neither 0 nor ZERO WIDTH SPACE are available\\%
3938      in the current font, and therefore the hyphen\\%
3939      will be printed. Try changing the fontspec's\\%
3940      'HyphenChar' to another value, but be aware\\%
3941      this setting is not safe (see the manual)}}%
3942   \hyphenchar\font\defaultshyphenchar
3943 \fi\fi
3944 \fi}%
3945 {\hyphenchar\font\defaultshyphenchar}}
3946 % \fi}

```

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).

```

3947 \def\bbl@ini@basic#1{%
3948   \def\BabelBeforeIni##1##2{%
3949     \begingroup
3950     \bbl@add\bbl@secpost@identification{\closein\bbl@readstream}%
3951     \bbl@read@ini{##1}1%
3952     \endinput           % babel- .tex may contain onlypreamble's
3953     \endgroup}%         boxed, to avoid extra spaces:
3954   {\bbl@input@texini{#1}}}

```

A tool to define the macros for native digits from the list provided in the ini file. Somewhat convoluted because there are 10 digits, but only 9 arguments in  $\TeX$ . Non-digits characters are kept. The first macro is the generic “localized” command.

```

3955 \def\bbl@setdigits#1#2#3#4#5{%
3956   \bbl@exp{%
3957     \def<\language digits>####1{%      ie, \langdigits
3958       \<\bbl@digits@\language>####1\\\@nil}%
3959       \let\<\bbl@cntr@digits@\language>\<\language digits>%
3960       \def<\language counter>####1{%    ie, \langcounter
3961         \\\expandafter\<\bbl@counter@\language>%
3962         \\\csname c@####1\endcsname}%
3963       \def\<\bbl@counter@\language>####1{% ie, \bbl@counter@lang
3964         \\\expandafter\<\bbl@digits@\language>%
3965         \\\number####1\\\@nil}}}%
3966   \def\bbl@tempa##1##2##3##4##5{%
3967     \bbl@exp{%      Wow, quite a lot of hashes! :-(
3968       \def\<\bbl@digits@\language>#####1{%
3969         \\\ifx#####1\\\@nil           % ie, \bbl@digits@lang
3970         \\\else
3971           \\\ifx0#####1#1%
3972           \\\else\\\ifx1#####1#2%
3973           \\\else\\\ifx2#####1#3%
3974           \\\else\\\ifx3#####1#4%
3975           \\\else\\\ifx4#####1#5%
3976           \\\else\\\ifx5#####1#1%
3977           \\\else\\\ifx6#####1#2%
3978           \\\else\\\ifx7#####1#3%

```





```

4027             The corresponding ini file has not been loaded\\%
4028             Perhaps it doesn't exist}%
4029             {See the manual for details.}}%
4030     {\bbl@cs{\csname bbl@info@#1\endcsname @\languagename}}}%
4031 % \@namedef{bbl@info@name.locale}{lcnname}
4032 \@namedef{bbl@info@tag.ini}{lini}
4033 \@namedef{bbl@info@name.english}{elname}
4034 \@namedef{bbl@info@name.opentype}{lname}
4035 \@namedef{bbl@info@tag.bcp47}{tbcpr}
4036 \@namedef{bbl@info@language.tag.bcp47}{lbcpr}
4037 \@namedef{bbl@info@tag.opentype}{lotf}
4038 \@namedef{bbl@info@script.name}{esname}
4039 \@namedef{bbl@info@script.name.opentype}{sname}
4040 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
4041 \@namedef{bbl@info@script.tag.opentype}{sotf}
4042 \let\bbl@ensureinfo\@gobble
4043 \newcommand\BabelEnsureInfo{%
4044   \ifx\InputIfFileExists\@undefined\else
4045     \def\bbl@ensureinfo##1{%
4046       \bbl@ifunset{bbl@lname@##1}{\bbl@ini@basic{##1}}{}}%
4047   \fi
4048   \bbl@foreach\bbl@loaded{%
4049     \def\languagename{##1}%
4050     \bbl@ensureinfo{##1}}}

```

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`.

```

4051 \newcommand\getlocaleproperty{%
4052   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
4053 \def\bbl@getproperty@s#1#2#3{%
4054   \let#1\relax
4055   \def\bbl@elt##1##2##3{%
4056     \bbl@ifsamestring{##1/##2}{##3}%
4057     {\providecommand#1{##3}%
4058     \def\bbl@elt####1####2####3{}}}%
4059   {}}%
4060   \bbl@cs{inidata@#2}}%
4061 \def\bbl@getproperty@x#1#2#3{%
4062   \bbl@getproperty@s{#1}{#2}{#3}%
4063   \ifx#1\relax
4064     \bbl@error
4065     {Unknown key for locale '#2':\%
4066     #3\%
4067     \string#1 will be set to \relax}%
4068     {Perhaps you misspelled it.}%
4069   \fi}
4070 \let\bbl@ini@loaded\@empty
4071 \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.

```

4072 \newcommand\babeladjust[1]{% TODO. Error handling.
4073   \bbl@forkv{#1}{%
4074     \bbl@ifunset{bbl@ADJ@##1@##2}%
4075     {\bbl@cs{ADJ@##1}{##2}}%
4076     {\bbl@cs{ADJ@##1@##2}}}

```

```

4077 %
4078 \def\bbl@adjust@lua#1#2{%
4079   \ifvmode
4080     \ifnum\currentgrouplevel=\z@
4081       \directlua{ Babel.#2 }%
4082       \expandafter\expandafter\expandafter\@gobble
4083     \fi
4084   \fi
4085   {\bbl@error   % The error is gobbled if everything went ok.
4086     {Currently, #1 related features can be adjusted only\\%
4087       in the main vertical list.}%
4088     {Maybe things change in the future, but this is what it is.}}}
4089 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
4090   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
4091 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
4092   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
4093 \@namedef{bbl@ADJ@bidi.text@on}{%
4094   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
4095 \@namedef{bbl@ADJ@bidi.text@off}{%
4096   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
4097 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
4098   \bbl@adjust@lua{bidi}{digits_mapped=true}}
4099 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
4100   \bbl@adjust@lua{bidi}{digits_mapped=false}}
4101 %
4102 \@namedef{bbl@ADJ@linebreak.sea@on}{%
4103   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
4104 \@namedef{bbl@ADJ@linebreak.sea@off}{%
4105   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
4106 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
4107   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
4108 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
4109   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
4110 %
4111 \def\bbl@adjust@layout#1{%
4112   \ifvmode
4113     #1%
4114     \expandafter\@gobble
4115   \fi
4116   {\bbl@error   % The error is gobbled if everything went ok.
4117     {Currently, layout related features can be adjusted only\\%
4118       in vertical mode.}%
4119     {Maybe things change in the future, but this is what it is.}}}
4120 \@namedef{bbl@ADJ@layout.tabular@on}{%
4121   \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}}
4122 \@namedef{bbl@ADJ@layout.tabular@off}{%
4123   \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}}
4124 \@namedef{bbl@ADJ@layout.lists@on}{%
4125   \bbl@adjust@layout{\let\list\bbl@NL@list}}
4126 \@namedef{bbl@ADJ@layout.lists@off}{%
4127   \bbl@adjust@layout{\let\list\bbl@OL@list}}
4128 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
4129   \bbl@activateposthyphen}
4130 %
4131 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
4132   \bbl@bcpallowedtrue}
4133 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
4134   \bbl@bcpallowedfalse}
4135 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%

```

```

4136 \def\bbl@bcp@prefix{#1}}
4137 \def\bbl@bcp@prefix{bcp47-}
4138 \@namedef{bbl@ADJ@autoload.options}#1{%
4139 \def\bbl@autoload@options{#1}}
4140 \let\bbl@autoload@bcptions\@empty
4141 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
4142 \def\bbl@autoload@bcptions{#1}}
4143 \newif\ifbbl@bcptoname
4144 \@namedef{bbl@ADJ@bcp47.toname@on}{%
4145 \bbl@bcptonametrue
4146 \BabelEnsureInfo}
4147 \@namedef{bbl@ADJ@bcp47.toname@off}{%
4148 \bbl@bcptonamefalse}
4149 % TODO: use babel name, override
4150 %
4151 % As the final task, load the code for lua.
4152 %
4153 \ifx\directlua\@undefined\else
4154 \ifx\bbl@luapatterns\@undefined
4155 \input luababel.def
4156 \fi
4157 \fi
4158 </core>

A proxy file for switch.def

4159 <*kernel>
4160 \let\bbl@onlyswitch\@empty
4161 \input babel.def
4162 \let\bbl@onlyswitch\@undefined
4163 </kernel>
4164 <*patterns>

```

## 11 Loading hyphenation patterns

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

To make sure that  $\text{\LaTeX}$  2.09 executes the `\@begindocumenthook` we would want to alter `\begin{document}`, but as this done too often already, we add the new code at the front of `\@preamblecmds`. But we can only do that after it has been defined, so we add this piece of code to `\dump`.

This new definition starts by adding an instruction to write a message on the terminal and in the transcript file to inform the user of the preloaded hyphenation patterns.

Then everything is restored to the old situation and the format is dumped.

```

4165 <<Make sure ProvidesFile is defined>>
4166 \ProvidesFile{hyphen.cfg}[<<date>> <<version>> Babel hyphens]
4167 \xdef\bbl@format{\jobname}
4168 \def\bbl@version{<<version>>}
4169 \def\bbl@date{<<date>>}
4170 \ifx\AtBeginDocument\@undefined
4171 \def\@empty{}
4172 \let\orig@dump\dump
4173 \def\dump{%
4174 \ifx\@ztryfc\@undefined
4175 \else
4176 \toks0=\expandafter{\@preamblecmds}%
4177 \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
4178 \def\@begindocumenthook{}%

```

```

4179 \fi
4180 \let\dump\orig@dump\let\orig@dump\@undefined\dump}
4181 \fi
4182 <<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.

```

4183 \def\process@line#1#2 #3 #4 {%
4184 \ifx=#1%
4185 \process@synonym{#2}%
4186 \else
4187 \process@language{#1#2}{#3}{#4}%
4188 \fi
4189 \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.

```

4190 \toks@{}
4191 \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.

```

4192 \def\process@synonym#1{%
4193 \ifnum\last@language=\m@ne
4194 \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4195 \else
4196 \expandafter\chardef\csname l@#1\endcsname\last@language
4197 \wlog{\string\l@#1=\string\language\the\last@language}%
4198 \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4199 \csname\language\hyphenmins\endcsname
4200 \let\bbl@elt\relax
4201 \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4202 \fi}

```

`\process@language` The macro `\process@language` is used to process a non-empty line from the ‘configuration file’. It has three arguments, each delimited by white space. The first argument is the ‘name’ of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions.

The first thing to do is call `\addlanguage` to allocate a pattern register and to make that register ‘active’. Then the pattern file is read.

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

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

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

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

When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does

not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

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

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

```

4203 \def\process@language#1#2#3{%
4204   \expandafter\addlanguage\csname l@#1\endcsname
4205   \expandafter\language\csname l@#1\endcsname
4206   \edef\language{#1}%
4207   \bbl@hook@everylanguage{#1}%
4208   % > luatex
4209   \bbl@get@enc#1::\@@@
4210   \begingroup
4211     \lefthyphenmin\m@ne
4212     \bbl@hook@loadpatterns{#2}%
4213     % > luatex
4214     \ifnum\lefthyphenmin=\m@ne
4215     \else
4216       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4217         \the\lefthyphenmin\the\righthyphenmin}%
4218     \fi
4219   \endgroup
4220   \def\bbl@tempa{#3}%
4221   \ifx\bbl@tempa\@empty\else
4222     \bbl@hook@loadexceptions{#3}%
4223     % > luatex
4224   \fi
4225   \let\bbl@elt\relax
4226   \edef\bbl@languages{%
4227     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4228   \ifnum\the\language=\z@
4229     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4230       \set@hyphenmins\tw@\thr@@\relax
4231     \else
4232       \expandafter\expandafter\expandafter\set@hyphenmins
4233       \csname #1hyphenmins\endcsname
4234     \fi
4235     \the\toks@
4236     \toks@{}%
4237   \fi}

```

\bbl@get@enc The macro \bbl@get@enc extracts the font encoding from the language name and stores it in  
\bbl@hyph@enc \bbl@hyph@enc. It uses delimited arguments to achieve this.

```

4238 \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.

```

4239 \def\bbl@hook@everylanguage#1{}
4240 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4241 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4242 \def\bbl@hook@loadkernel#1{%
4243   \def\addlanguage{\csname newlanguage\endcsname}%
4244   \def\adddialect##1##2{%
4245     \global\chardef##1##2\relax
4246     \wlog{\string##1 = a dialect from \string\language##2}}%

```

```

4247 \def\iflanguage##1{%
4248   \expandafter\ifx\csname l@##1\endcsname\relax
4249     \@nolanerr{##1}%
4250   \else
4251     \ifnum\csname l@##1\endcsname=\language
4252       \expandafter\expandafter\expandafter\@firstoftwo
4253     \else
4254       \expandafter\expandafter\expandafter\@secondoftwo
4255     \fi
4256   \fi}%
4257 \def\providehyphenmins##1##2{%
4258   \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4259     \@namedef{##1hyphenmins}{##2}%
4260   \fi}%
4261 \def\set@hyphenmins##1##2{%
4262   \lefthyphenmin##1\relax
4263   \righthyphenmin##2\relax}%
4264 \def\selectlanguage{%
4265   \errhelp{Selecting a language requires a package supporting it}%
4266   \errmessage{Not loaded}}%
4267 \let\foreignlanguage\selectlanguage
4268 \let\otherlanguage\selectlanguage
4269 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4270 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4271 \def\setlocale{%
4272   \errhelp{Find an armchair, sit down and wait}%
4273   \errmessage{Not yet available}}%
4274 \let\uselocale\setlocale
4275 \let\locale\setlocale
4276 \let\selectlocale\setlocale
4277 \let\localename\setlocale
4278 \let\textlocale\setlocale
4279 \let\textlanguage\setlocale
4280 \let\languagetext\setlocale}
4281 \begingroup
4282 \def\AddBabelHook#1#2{%
4283   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4284     \def\next{\toks1}%
4285   \else
4286     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4287   \fi
4288   \next}
4289 \ifx\directlua\undefined
4290   \ifx\XeTeXinputencoding\undefined\else
4291     \input xebabel.def
4292   \fi
4293 \else
4294   \input luababel.def
4295 \fi
4296 \openin1 = babel-\bbl@format.cfg
4297 \ifeof1
4298 \else
4299   \input babel-\bbl@format.cfg\relax
4300 \fi
4301 \closein1
4302 \endgroup
4303 \bbl@hook@loadkernel{switch.def}

```

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

```
4304 \openin1 = language.dat
```

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

```
4305 \def\language{english}%
4306 \ifeof1
4307   \message{I couldn't find the file language.dat,\space
4308             I will try the file hyphen.tex}
4309   \input hyphen.tex\relax
4310   \chardef\l@english\z@
4311 \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  $-1$ .

```
4312   \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.

```
4313   \loop
4314     \endlinechar\m@ne
4315     \read1 to \bbl@line
4316     \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.

```
4317     \if T\ifeof1F\fi T\relax
4318     \ifx\bbl@line\@empty\else
4319       \edef\bbl@line{\bbl@line\space\space\space}%
4320       \expandafter\process@line\bbl@line\relax
4321     \fi
4322   \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.

```
4323   \begingroup
4324     \def\bbl@elt#1#2#3#4{%
4325       \global\language=#2\relax
4326       \gdef\language{#1}%
4327       \def\bbl@elt##1##2##3##4{}}%
4328     \bbl@languages
4329   \endgroup
4330 \fi
4331 \closein1
```

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

```
4332 \if/\the\toks@/\else
4333   \errhelp{language.dat loads no language, only synonyms}
4334   \errmessage{Orphan language synonym}
4335 \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.

```
4336 \let\bbl@line\@undefined
4337 \let\process@line\@undefined
4338 \let\process@synonym\@undefined
```



```

4339 \let\process@language\@undefined
4340 \let\bbl@get@enc\@undefined
4341 \let\bbl@hyph@enc\@undefined
4342 \let\bbl@tempa\@undefined
4343 \let\bbl@hook@loadkernel\@undefined
4344 \let\bbl@hook@everylanguage\@undefined
4345 \let\bbl@hook@loadpatterns\@undefined
4346 \let\bbl@hook@loadexceptions\@undefined
4347 \end{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].

```

4348 <<*More package options>> ≡
4349 \chardef\bbl@bidimode\z@
4350 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4351 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4352 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4353 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4354 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4355 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4356 <</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`.

At the time of this writing, `fontspec` shows a warning about there are languages not available, which some people think refers to `babel`, even if there is nothing wrong. Here is hack to patch `fontspec` to avoid the misleading message, which is replaced by a more explanatory one.

```

4357 <<*Font selection>> ≡
4358 \bbl@trace{Font handling with fontspec}
4359 \ifx\ExplSyntaxOn\@undefined\else
4360   \ExplSyntaxOn
4361   \catcode`\ =10
4362   \def\bbl@loadfontspec{%
4363     \usepackage{fontspec}%
4364     \expandafter
4365     \def\csname msg-text~>~fontspec/language-not-exist\endcsname##1##2##3##4{%
4366       Font '\l_fontspec_fontname_tl' is using the\\%
4367       default features for language '##1'.\\%
4368       That's usually fine, because many languages\\%
4369       require no specific features, but if the output is\\%
4370       not as expected, consider selecting another font.}
4371     \expandafter
4372     \def\csname msg-text~>~fontspec/no-script\endcsname##1##2##3##4{%
4373       Font '\l_fontspec_fontname_tl' is using the\\%
4374       default features for script '##2'.\\%
4375       That's not always wrong, but if the output is\\%
4376       not as expected, consider selecting another font.}}
4377   \ExplSyntaxOff
4378 \fi
4379 \@onlypreamble\babelfont
4380 \newcommand\babelfont[2][\% 1=langs/scripts 2=fam
4381   \bbl@foreach{#1}{\%
4382     \expandafter\ifx\csname date##1\endcsname\relax

```

```

4383 \IfFileExists{babel-##1.tex}%
4384 {\babelprovide{##1}}%
4385 {}%
4386 \fi}%
4387 \edef\bbl@tempa{#1}%
4388 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4389 \ifx\fontspec\undefined
4390 \bbl@loadfontspec
4391 \fi
4392 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4393 \bbl@bblfont}
4394 \newcommand\bbl@bblfont[2][{}% 1=features 2=fontname, @font=rm|sf|tt
4395 \bbl@ifunset{\bbl@tempb family}%
4396 {\bbl@providefam{\bbl@tempb}}%
4397 {\bbl@exp{%
4398 \\\bbl@sreplace\<\bbl@tempb family >%
4399 {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
4400 % For the default font, just in case:
4401 \bbl@ifunset{\bbl@lsys@language}{\bbl@provide@lsys{\language}}}%
4402 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4403 {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
4404 \bbl@exp{%
4405 \let\<\bbl@tempb dflt@\language>\<\bbl@tempb dflt@>%
4406 \\\bbl@font@set\<\bbl@tempb dflt@\language>%
4407 \<\bbl@tempb default>\<\bbl@tempb family>}}%
4408 {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4409 \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:

```

4410 \def\bbl@providefam#1{%
4411 \bbl@exp{%
4412 \\\newcommand\<#1default>{}% Just define it
4413 \\\bbl@add@list\\bbl@font@fams{#1}%
4414 \\\DeclareRobustCommand\<#1family>{%
4415 \\\not@math@alphabet\<#1family>\relax
4416 \\\fontfamily\<#1default>\selectfont}%
4417 \\\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.

```

4418 \def\bbl@nostdfont#1{%
4419 \bbl@ifunset{\bbl@WFF@f@family}%
4420 {\bbl@csarg\gdef{\bbl@WFF@f@family}}}% Flag, to avoid dupl warns
4421 \bbl@infowarn{The current font is not a babel standard family:\\%
4422 #1%
4423 \fontname\font\\%
4424 There is nothing intrinsically wrong with this warning, and\\%
4425 you can ignore it altogether if you do not need these\\%
4426 families. But if they are used in the document, you should be\\%
4427 aware 'babel' will no set Script and Language for them, so\\%
4428 you may consider defining a new family with \string\babelfont.\\%
4429 See the manual for further details about \string\babelfont.\\%
4430 Reported}}
4431 {}}%
4432 \gdef\bbl@switchfont{%
4433 \bbl@ifunset{\bbl@lsys@language}{\bbl@provide@lsys{\language}}}%
4434 \bbl@exp{% eg Arabic -> arabic
4435 \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}}%
4436 \bbl@foreach\bbl@font@fams{%

```

```

4437 \bbl@ifunset{\bbl@##1dflt@\language}% (1) language?
4438 {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}% (2) from script?
4439 {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
4440 {}% 123=F - nothing!
4441 {\bbl@exp{% 3=T - from generic
4442 \global\let\<\bbl@##1dflt@\language>%
4443 \<\bbl@##1dflt@>}}}%
4444 {\bbl@exp{% 2=T - from script
4445 \global\let\<\bbl@##1dflt@\language>%
4446 \<\bbl@##1dflt@*\bbl@tempa>}}}%
4447 {}% 1=T - language, already defined
4448 \def\bbl@tempa{\bbl@nostdfont{}}}%
4449 \bbl@foreach\bbl@font@fams{% don't gather with prev for
4450 \bbl@ifunset{\bbl@##1dflt@\language}%
4451 {\bbl@cs{famrst@##1}%
4452 \global\bbl@csarg\let{famrst@##1}\relax}%
4453 {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4454 \\bbl@add\\originalTeX{%
4455 \\bbl@font@rst{\bbl@cl{##1dflt}}}%
4456 \<##1default>\<##1family>{##1}}}%
4457 \\bbl@font@set\<\bbl@##1dflt@\language>% the main part!
4458 \<##1default>\<##1family>}}}%
4459 \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.

```

4460 \ifx\fontfamily\undefined\else % if latex
4461 \ifcase\bbl@engine % if pdftex
4462 \let\bbl@ckeckstdfonts\relax
4463 \else
4464 \def\bbl@ckeckstdfonts{%
4465 \begingroup
4466 \global\let\bbl@ckeckstdfonts\relax
4467 \let\bbl@tempa\@empty
4468 \bbl@foreach\bbl@font@fams{%
4469 \bbl@ifunset{\bbl@##1dflt@}%
4470 {\nameuse{##1family}%
4471 \bbl@csarg\gdef{WFF@f@family}}}% Flag
4472 \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= \fontfamily\\
4473 \space\space\fontname\font\\}}}%
4474 \bbl@csarg\xdef{##1dflt@}{\fontfamily}%
4475 \expandafter\xdef\csname ##1default\endcsname{\fontfamily}}}%
4476 {}}%
4477 \ifx\bbl@tempa\@empty\else
4478 \bbl@infowarn{The following font families will use the default\\
4479 settings for all or some languages:\\
4480 \bbl@tempa
4481 There is nothing intrinsically wrong with it, but\\
4482 'babel' will no set Script and Language, which could\\
4483 be relevant in some languages. If your document uses\\
4484 these families, consider redefining them with \string\babelfont.\\
4485 Reported}%
4486 \fi
4487 \endgroup}
4488 \fi
4489 \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.

```
4490 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4491 \bbl@xin@{<>}{#1}%
4492 \fin@
4493 \bbl@exp{\bbl@fontspec@set\#1\expandafter@gobbletwo#1\#3}%
4494 \fi
4495 \bbl@exp{% 'Unprotected' macros return prev values
4496 \def\#2#1% eg, \rmdefault{\bbl@rmdflt@lang}
4497 \bbl@ifsamestring{#2}{\f@family}%
4498 {\#3%
4499 \bbl@ifsamestring{\f@series}{\bfdefault}{\bfseries}{}}%
4500 \let\bbl@tempa\relax}%
4501 {}}
4502 % TODO - next should be global?, but even local does its job. I'm
4503 % still not sure -- must investigate:
4504 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4505 \let\bbl@tempe\bbl@mapselect
4506 \let\bbl@mapselect\relax
4507 \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4508 \let#4@empty % Make sure \renewfontfamily is valid
4509 \bbl@exp{%
4510 \let\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily'
4511 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4512 {\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4513 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4514 {\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4515 \renewfontfamily\#4%
4516 [\bbl@cs{lsys@\languagename},#2]{#3}% ie \bbl@exp{.}{#3}
4517 \begingroup
4518 #4%
4519 \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4520 \endgroup
4521 \let#4\bbl@temp@fam
4522 \bbl@exp{\let\<\bbl@stripslash#4\space>\bbl@temp@pfam
4523 \let\bbl@mapselect\bbl@tempe}%
```

`font@rst` and `famrst` are only used when there is no global settings, to save and restore the previous families. Not really necessary, but done for optimization.

```
4524 \def\bbl@font@rst#1#2#3#4{%
4525 \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`.

```
4526 \def\bbl@font@fams{rm,sf,tt}
```

The old tentative way. Short and preserved for compatibility, but deprecated. Note there is no direct alternative for `\babelFSfeatures`. The reason is explained in the user guide, but essentially – that was not the way to go :-).

```
4527 \newcommand\babelFSstore[2][{%
4528 \bbl@ifblank{#1}%
4529 {\bbl@csarg\def{sname@#2}{Latin}}%
4530 {\bbl@csarg\def{sname@#2}{#1}}%
4531 \bbl@provide@dirs{#2}%
4532 \bbl@csarg\ifnum{wdir@#2}>\z@
4533 \let\bbl@beforeforeign\leavevmode
4534 \EnableBabelHook{babel-bidi}%
4535 \fi
4536 \bbl@foreach{#2}{%
4537 \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4538 \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
```

```

4539 \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4540 \def\bbl@FSstore#1#2#3#4{%
4541 \bbl@csarg\edef{#2default#1}{#3}%
4542 \expandafter\addto\csname extras#1\endcsname{%
4543 \let#4#3%
4544 \ifx#3\f@family
4545 \edef#3{\csname bbl@#2default#1\endcsname}%
4546 \fontfamily{#3}\selectfont
4547 \else
4548 \edef#3{\csname bbl@#2default#1\endcsname}%
4549 \fi}%
4550 \expandafter\addto\csname noextras#1\endcsname{%
4551 \ifx#3\f@family
4552 \fontfamily{#4}\selectfont
4553 \fi
4554 \let#3#4}}
4555 \let\bbl@langfeatures\empty
4556 \def\babelFSfeatures{% make sure \fontspec is redefined once
4557 \let\bbl@ori@fontspec\fontspec
4558 \renewcommand\fontspec[1][{}]{%
4559 \bbl@ori@fontspec[\bbl@langfeatures##1]}
4560 \let\babelFSfeatures\bbl@FSfeatures
4561 \babelFSfeatures}
4562 \def\bbl@FSfeatures#1#2{%
4563 \expandafter\addto\csname extras#1\endcsname{%
4564 \babel@save\bbl@langfeatures
4565 \edef\bbl@langfeatures{#2,}}
4566 <</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.

```

4567 <<(*Footnote changes)>> ≡
4568 \bbl@trace{Bidi footnotes}
4569 \ifnum\bbl@bidimode>\z@
4570 \def\bbl@footnote#1#2#3{%
4571 \ifnextchar[%
4572 {\bbl@footnote@o{#1}{#2}{#3}}%
4573 {\bbl@footnote@x{#1}{#2}{#3}}}
4574 \long\def\bbl@footnote@x#1#2#3#4{%
4575 \bgroup
4576 \select@language@x{\bbl@main@language}%
4577 \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4578 \egroup}
4579 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4580 \bgroup
4581 \select@language@x{\bbl@main@language}%
4582 \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4583 \egroup}
4584 \def\bbl@footnotetext#1#2#3{%
4585 \ifnextchar[%
4586 {\bbl@footnotetext@o{#1}{#2}{#3}}%
4587 {\bbl@footnotetext@x{#1}{#2}{#3}}}
4588 \long\def\bbl@footnotetext@x#1#2#3#4{%
4589 \bgroup

```

```

4590 \select@language@x{\bbl@main@language}%
4591 \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4592 \egroup}
4593 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4594 \bgroup
4595 \select@language@x{\bbl@main@language}%
4596 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4597 \egroup}
4598 \def\BabelFootnote#1#2#3#4{%
4599 \ifx\bbl@fn@footnote\@undefined
4600 \let\bbl@fn@footnote\footnote
4601 \fi
4602 \ifx\bbl@fn@footnotetext\@undefined
4603 \let\bbl@fn@footnotetext\footnotetext
4604 \fi
4605 \bbl@ifblank{#2}%
4606 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4607 \namedef{\bbl@stripslash#1text}%
4608 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4609 {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4610 \namedef{\bbl@stripslash#1text}%
4611 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4612 \fi
4613 <</Footnote changes>>

```

Now, the code.

```

4614 (*xetex)
4615 \def\BabelStringsDefault{unicode}
4616 \let\xebbl@stop\relax
4617 \AddBabelHook{xetex}{encodedcommands}{%
4618 \def\bbl@tempa{#1}%
4619 \ifx\bbl@tempa\@empty
4620 \XeTeXinputencoding"bytes"%
4621 \else
4622 \XeTeXinputencoding"#1"%
4623 \fi
4624 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4625 \AddBabelHook{xetex}{stopcommands}{%
4626 \xebbl@stop
4627 \let\xebbl@stop\relax}
4628 \def\bbl@intraspace#1 #2 #3\@@{%
4629 \bbl@csarg\gdef{\xeisp@{language}}{
4630 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4631 \def\bbl@intrapenalty#1\@@{%
4632 \bbl@csarg\gdef{\xeipn@{language}}{
4633 {\XeTeXlinebreakpenalty #1\relax}}
4634 \def\bbl@provide@intraspace{%
4635 \bbl@xin@{\bbl@cl{lnbrk}}{s}%
4636 \ifin@else\bbl@xin@{\bbl@cl{lnbrk}}{c}\fi
4637 \ifin@
4638 \bbl@ifunset{bbl@intsp@{language}}{%
4639 {\expandafter\ifx\csname bbl@intsp@{language}\endcsname\@empty\else
4640 \ifx\bbl@KVP@intraspace\@nil
4641 \bbl@exp{%
4642 \bbl@intraspace\bbl@cl{intsp}\@@}%
4643 \fi
4644 \ifx\bbl@KVP@intrapenalty\@nil
4645 \bbl@intrapenalty0\@@
4646 \fi

```

```

4647 \fi
4648 \ifx\bb1@KVP@intraspace\@nil\else % We may override the ini
4649 \expandafter\bb1@intraspace\bb1@KVP@intraspace\@@
4650 \fi
4651 \ifx\bb1@KVP@intrapenalty\@nil\else
4652 \expandafter\bb1@intrapenalty\bb1@KVP@intrapenalty\@@
4653 \fi
4654 \bb1@exp{%
4655 \\\bb1@add\<extras\language>{%
4656 \XeTeXlinebreaklocale "\bb1@cl{tbc}"%
4657 \<bb1@xeisp@\language>%
4658 \<bb1@xeipn@\language>}%
4659 \\\bb1@tglobal\<extras\language>%
4660 \\\bb1@add\<noextras\language>{%
4661 \XeTeXlinebreaklocale "en"%
4662 \\\bb1@tglobal\<noextras\language>}%
4663 \ifx\bb1@ispace\@undefined
4664 \gdef\bb1@ispace{\bb1@cl{xeisp}}%
4665 \ifx\AtBeginDocument\@notprerr
4666 \expandafter\@secondoftwo % to execute right now
4667 \fi
4668 \AtBeginDocument{%
4669 \expandafter\bb1@add
4670 \csname selectfont \endcsname{\bb1@ispace}%
4671 \expandafter\bb1@tglobal\csname selectfont \endcsname}%
4672 \fi}%
4673 \fi}
4674 \ifx\DisableBabelHook\@undefined\endinput\fi
4675 \AddBabelHook{babel-fontspec}{afterextras}{\bb1@switchfont}
4676 \AddBabelHook{babel-fontspec}{beforestart}{\bb1@ckeckstdfonts}
4677 \DisableBabelHook{babel-fontspec}
4678 <<Font selection>>
4679 \input txtbabel.def
4680 </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.

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

\advance\bb1@startskip\adim, \bb1@startskip\adim.

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

```

4681 <*texxet>
4682 \providecommand\bb1@provide@intraspace{}
4683 \bb1@trace{Redefinitions for bidi layout}
4684 \def\bb1@sspre@caption{%
4685 \bb1@exp{\everyhbox{\\\bb1@textdir\bb1@cs{wdir}\bb1@main@language}}}}
4686 \ifx\bb1@opt@layout\@nnil\endinput\fi % No layout
4687 \def\bb1@startskip{\ifcase\bb1@thepardir\leftskip\else\rightskip\fi}
4688 \def\bb1@endskip{\ifcase\bb1@thepardir\rightskip\else\leftskip\fi}
4689 \ifx\bb1@beforeforeign\leavevmode % A poor test for bidi=
4690 \def\@hangfrom#1{%
4691 \setbox\@tempboxa\hbox{#1}}%
4692 \hangindent\ifcase\bb1@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4693 \noindent\box\@tempboxa}
4694 \def\raggedright{%

```

```

4695 \let\\@centercr
4696 \bbl@startskip\z@skip
4697 \@rightskip\@flushglue
4698 \bbl@endskip\@rightskip
4699 \parindent\z@
4700 \parfillskip\bbl@startskip}
4701 \def\raggedleft{%
4702 \let\\@centercr
4703 \bbl@startskip\@flushglue
4704 \bbl@endskip\z@skip
4705 \parindent\z@
4706 \parfillskip\bbl@endskip}
4707 \fi
4708 \IfBabelLayout{lists}
4709 {\bbl@sreplace\list
4710 {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4711 \def\bbl@listleftmargin{%
4712 \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4713 \ifcase\bbl@engine
4714 \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4715 \def\p@enumiii{\p@enumii}\theenumii}%
4716 \fi
4717 \bbl@sreplace\@verbatim
4718 {\leftskip\@totalleftmargin}%
4719 {\bbl@startskip\textwidth
4720 \advance\bbl@startskip-\linewidth}%
4721 \bbl@sreplace\@verbatim
4722 {\rightskip\z@skip}%
4723 {\bbl@endskip\z@skip}}%
4724 {}
4725 \IfBabelLayout{contents}
4726 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4727 \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4728 {}
4729 \IfBabelLayout{columns}
4730 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4731 \def\bbl@outputbox#1{%
4732 \hb@xt@\textwidth{%
4733 \hskip\columnwidth
4734 \hfil
4735 {\normalcolor\vrule \@width\columnseprule}%
4736 \hfil
4737 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4738 \hskip-\textwidth
4739 \hb@xt@\columnwidth{\box\@outputbox \hss}%
4740 \hskip\columnsep
4741 \hskip\columnwidth}}}%
4742 {}
4743 <<Footnote changes>>
4744 \IfBabelLayout{footnotes}%
4745 {\BabelFootnote\footnote\language\{}}%
4746 \BabelFootnote\localfootnote\language\{}}%
4747 \BabelFootnote\mainfootnote\{}}%
4748 {}

```

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.

```

4749 \IfBabelLayout{counters}%
4750 {\let\bbl@latinarabic=\@arabic

```



```

4751 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4752 \let\bbl@asciroman=\@roman
4753 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4754 \let\bbl@asciiRoman=\@Roman
4755 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}{}}
4756 \end{texet}

```

### 13.3 LuaTeX

The loader for luatex is based solely on language.dat, which is read on the fly. The code shouldn't be executed when the format is build, so we check if \AddBabelHook is defined. Then comes a modified version of the loader in hyphen.cfg (without the hyphenmins stuff, which is under the direct control of babel).

The names \l@<language> are defined and take some value from the beginning because all ldf files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first selected (which usually means when the ldf finishes). If a language has been loaded, \bbl@hyphendata@<num> exists (with the names of the files read).

The default setup preloads the first language into the format. This is intended mainly for 'english', so that it's available without further intervention from the user. To avoid duplicating it, the following rule applies: if the "0th" language and the first language in language.dat have the same name then just ignore the latter. If there are new synonymous, they are added, but note if the language patterns have not been preloaded they won't at run time.

Other preloaded languages could be read twice, if they have been preloaded into the format. This is not optimal, but it shouldn't happen very often – with luatex patterns are best loaded when the document is typeset, and the "0th" language is preloaded just for backwards compatibility.

As of 1.1b, lua(e)tex is taken into account. Formerly, loading of patterns on the fly didn't work in this format, but with the new loader it does. Unfortunately, the format is not based on babel, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format language.dat is used (under the principle of a single source), instead of language.def.

Of course, there is room for improvements, like tools to read and reassign languages, which would require modifying the language list, and better error handling.

We need catcode tables, but no format (targeted by babel) provide a command to allocate them (although there are packages like ctablestack). FIX - This isn't true anymore. For the moment, a dangerous approach is used - just allocate a high random number and cross the fingers. To complicate things, etex.sty changes the way languages are allocated.

This files is read at three places: (1) when plain.def, babel.sty starts, to read the list of available languages from language.dat (for the base option); (2) at hyphen.cfg, to modify some macros; (3) in the middle of plain.def and babel.sty, by babel.def, with the commands and other definitions for luatex (eg. \babelpatterns).

```

4757 (*luatex)
4758 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4759 \bbl@trace{Read language.dat}
4760 \ifx\bbl@readstream\undefined
4761 \csname newread\endcsname\bbl@readstream
4762 \fi
4763 \begingroup
4764 \toks@{}
4765 \count@ \z@ % 0=start, 1=0th, 2=normal
4766 \def\bbl@process@line#1#2 #3 #4 {%
4767 \ifx=#1%
4768 \bbl@process@synonym{#2}%
4769 \else
4770 \bbl@process@language{#1#2}{#3}{#4}%
4771 \fi
4772 \ignorespaces}
4773 \def\bbl@manylang{%

```

```

4774 \ifnum\bb1@last>\@ne
4775 \bb1@info{Non-standard hyphenation setup}%
4776 \fi
4777 \let\bb1@manylang\relax}
4778 \def\bb1@process@language#1#2#3{%
4779 \ifcase\count@
4780 \@ifundefined{zth@#1}{\count@tw@}{\count@ne}%
4781 \or
4782 \count@tw@
4783 \fi
4784 \ifnum\count@=tw@
4785 \expandafter\addlanguage\csname l@#1\endcsname
4786 \language\allocationnumber
4787 \chardef\bb1@last\allocationnumber
4788 \bb1@manylang
4789 \let\bb1@elt\relax
4790 \xdef\bb1@languages{%
4791 \bb1@languages\bb1@elt{#1}{\the\language}{#2}{#3}}%
4792 \fi
4793 \the\toks@
4794 \toks@{}}
4795 \def\bb1@process@synonym@aux#1#2{%
4796 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4797 \let\bb1@elt\relax
4798 \xdef\bb1@languages{%
4799 \bb1@languages\bb1@elt{#1}{#2}{}}}%
4800 \def\bb1@process@synonym#1{%
4801 \ifcase\count@
4802 \toks@\expandafter{\the\toks@\relax\bb1@process@synonym{#1}}%
4803 \or
4804 \@ifundefined{zth@#1}{\bb1@process@synonym@aux{#1}{0}}{%
4805 \else
4806 \bb1@process@synonym@aux{#1}{\the\bb1@last}%
4807 \fi}
4808 \ifx\bb1@languages@\undefined % Just a (sensible?) guess
4809 \chardef\l@english\z@
4810 \chardef\l@USenglish\z@
4811 \chardef\bb1@last\z@
4812 \global\@namedef{\bb1@hyphendata@0}{\hyphen.tex}}
4813 \gdef\bb1@languages{%
4814 \bb1@elt{english}{0}{\hyphen.tex}}%
4815 \bb1@elt{USenglish}{0}{}}
4816 \else
4817 \global\let\bb1@languages@format\bb1@languages
4818 \def\bb1@elt#1#2#3#4{% Remove all except language 0
4819 \ifnum#2>\z@\else
4820 \noexpand\bb1@elt{#1}{#2}{#3}{#4}%
4821 \fi}%
4822 \xdef\bb1@languages{\bb1@languages}%
4823 \fi
4824 \def\bb1@elt#1#2#3#4{\@namedef{zth@#1}} % Define flags
4825 \bb1@languages
4826 \openin\bb1@readstream=language.dat
4827 \ifeof\bb1@readstream
4828 \bb1@warning{I couldn't find language.dat. No additional\%
4829 patterns loaded. Reported}%
4830 \else
4831 \loop
4832 \endlinechar\m@ne

```

```

4833 \read\bbl@readstream to \bbl@line
4834 \endlinechar``^^M
4835 \if T\ifeof\bbl@readstream F\fi T\relax
4836 \ifx\bbl@line@empty\else
4837 \edef\bbl@line{\bbl@line\space\space\space}%
4838 \expandafter\bbl@process@line\bbl@line\relax
4839 \fi
4840 \repeat
4841 \fi
4842 \endgroup
4843 \bbl@trace{Macros for reading patterns files}
4844 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
4845 \ifx\babelcatcodetablenum\undefined
4846 \ifx\newcatcodetable\undefined
4847 \def\babelcatcodetablenum{5211}
4848 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4849 \else
4850 \newcatcodetable\babelcatcodetablenum
4851 \newcatcodetable\bbl@pattcodes
4852 \fi
4853 \else
4854 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4855 \fi
4856 \def\bbl@luapatterns#1#2{%
4857 \bbl@get@enc#1::\@@@
4858 \setbox\z@\hbox\bgroup
4859 \begingroup
4860 \savecatcodetable\babelcatcodetablenum\relax
4861 \initcatcodetable\bbl@pattcodes\relax
4862 \catcodetable\bbl@pattcodes\relax
4863 \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4864 \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
4865 \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4866 \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4867 \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4868 \catcode`\`=12 \catcode`\'=12 \catcode`\\"=12
4869 \input #1\relax
4870 \catcodetable\babelcatcodetablenum\relax
4871 \endgroup
4872 \def\bbl@tempa{#2}%
4873 \ifx\bbl@tempa@empty\else
4874 \input #2\relax
4875 \fi
4876 \egroup}%
4877 \def\bbl@patterns@lua#1{%
4878 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4879 \csname l@#1\endcsname
4880 \edef\bbl@tempa{#1}%
4881 \else
4882 \csname l@#1:\f@encoding\endcsname
4883 \edef\bbl@tempa{#1:\f@encoding}%
4884 \fi\relax
4885 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4886 \@ifundefined{bbl@hyphendata@the\language}%
4887 {\def\bbl@elt##1##2##3##4{%
4888 \ifnum##2=\csname l@#1:\f@encoding\endcsname % #2=spanish, dutch:OT1...
4889 \def\bbl@tempb{##3}%
4890 \ifx\bbl@tempb@empty\else % if not a synonymous
4891 \def\bbl@tempc{##3}{##4}}%

```

```

4892         \fi
4893         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4894     \fi}%
4895     \bbl@languages
4896     \@ifundefined{bbl@hyphendata@the\language}%
4897     {\bbl@info{No hyphenation patterns were set for\%
4898         language '\bbl@tempa'. Reported}}%
4899     {\expandafter\expandafter\expandafter\bbl@luapatterns
4900         \csname bbl@hyphendata@the\language\endcsname}}}%
4901 \endinput\fi
4902 % Here ends \ifx\AddBabelHook\@undefined
4903 % A few lines are only read by hyphen.cfg
4904 \ifx\DisableBabelHook\@undefined
4905     \AddBabelHook{luatex}{everylanguage}{%
4906         \def\process@language##1##2##3{%
4907             \def\process@line####1####2 ####3 ####4 {}}}%
4908     \AddBabelHook{luatex}{loadpatterns}{%
4909         \input #1\relax
4910         \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4911             {{#1}}}%
4912     \AddBabelHook{luatex}{loadexceptions}{%
4913         \input #1\relax
4914         \def\bbl@tempb##1##2{{#1}{#1}}}%
4915         \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
4916             {\expandafter\expandafter\expandafter\bbl@tempb
4917                 \csname bbl@hyphendata@the\language\endcsname}}%
4918 \endinput\fi
4919 % Here stops reading code for hyphen.cfg
4920 % The following is read the 2nd time it's loaded
4921 \begingroup % TODO - to a lua file
4922 \catcode`\%=12
4923 \catcode`\'=12
4924 \catcode`\%=12
4925 \catcode`\:=12
4926 \directlua{
4927     Babel = Babel or {}
4928     function Babel.bytes(line)
4929         return line:gsub(".",
4930             function (chr) return unicode.utf8.char(string.byte(chr)) end)
4931     end
4932     function Babel.begin_process_input()
4933         if luatexbase and luatexbase.add_to_callback then
4934             luatexbase.add_to_callback('process_input_buffer',
4935                 Babel.bytes, 'Babel.bytes')
4936         else
4937             Babel.callback = callback.find('process_input_buffer')
4938             callback.register('process_input_buffer', Babel.bytes)
4939         end
4940     end
4941     function Babel.end_process_input ()
4942         if luatexbase and luatexbase.remove_from_callback then
4943             luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
4944         else
4945             callback.register('process_input_buffer', Babel.callback)
4946         end
4947     end
4948     function Babel.addpatterns(pp, lg)
4949         local lg = lang.new(lg)
4950         local pats = lang.patterns(lg) or ''

```

```

4951 lang.clear_patterns(lg)
4952 for p in pp:gmatch('[^%s]+') do
4953     ss = ''
4954     for i in string.utfcharacters(p:gsub('%d', '')) do
4955         ss = ss .. '%d?' .. i
4956     end
4957     ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
4958     ss = ss:gsub('%.%%d%?$', '%%.')
4959     pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4960     if n == 0 then
4961         tex.sprint(
4962             [[\string\csname\space bbl@info\endcsname{New pattern: }]]
4963             .. p .. [[]])
4964         pats = pats .. ' ' .. p
4965     else
4966         tex.sprint(
4967             [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
4968             .. p .. [[]])
4969     end
4970 end
4971 lang.patterns(lg, pats)
4972 end
4973 }
4974 \endgroup
4975 \ifx\newattribute\undefined\else
4976 \newattribute\bbl@attr@locale
4977 \directlua{Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale'}
4978 \AddBabelHook{luatex}{beforeextras}{%
4979     \setattribute\bbl@attr@locale\localeid}
4980 \fi
4981 \def\BabelStringsDefault{unicode}
4982 \let\luabbl@stop\relax
4983 \AddBabelHook{luatex}{encodedcommands}{%
4984     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
4985     \ifx\bbl@tempa\bbl@tempb\else
4986         \directlua{Babel.begin_process_input()}%
4987         \def\luabbl@stop{%
4988             \directlua{Babel.end_process_input()}}%
4989     \fi}%
4990 \AddBabelHook{luatex}{stopcommands}{%
4991     \luabbl@stop
4992     \let\luabbl@stop\relax}
4993 \AddBabelHook{luatex}{patterns}{%
4994     \@ifundefined{bbl@hyphendata@the\language}%
4995     {\def\bbl@elt##1##2##3##4{%
4996         \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
4997         \def\bbl@tempb{##3}%
4998         \ifx\bbl@tempb\empty\else % if not a synonymous
4999             \def\bbl@tempc{##3}{##4}}%
5000         \fi
5001         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5002         \fi}%
5003     \bbl@languages
5004     \@ifundefined{bbl@hyphendata@the\language}%
5005     {\bbl@info{No hyphenation patterns were set for\%
5006         language '#2'. Reported}}%
5007     {\expandafter\expandafter\expandafter\bbl@luapatterns
5008         \csname bbl@hyphendata@the\language\endcsname}}}%
5009     \@ifundefined{bbl@patterns@}{}%

```

```

5010 \begingroup
5011 \bbl@xin@{, \number\language,}{, \bbl@pttnlist}%
5012 \ifin@else
5013 \ifx\bbl@patterns@\empty\else
5014 \directlua{ Babel.addpatterns(
5015 [[\bbl@patterns@]], \number\language) }%
5016 \fi
5017 \ifundefined{bbl@patterns@#1}%
5018 \empty
5019 {\directlua{ Babel.addpatterns(
5020 [[\space\csname bbl@patterns@#1\endcsname]],
5021 \number\language) }}%
5022 \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5023 \fi
5024 \endgroup}%
5025 \bbl@exp{%
5026 \bbl@ifunset{bbl@prehc@\languagename}{}%
5027 {\bbl@ifblank{\bbl@cs{prehc@\languagename}}{}}%
5028 {\prehyphenchar=\bbl@cl{prehc}\relax}}

```

**\babelpatterns** This macro adds patterns. Two macros are used to store them: `\bbl@patterns@` for the global ones and `\bbl@patterns@<lang>` for language ones. We make sure there is a space between words when multiple commands are used.

```

5029 \@onlypreamble\babelpatterns
5030 \AtEndOfPackage{%
5031 \newcommand\babelpatterns[2][\empty]{%
5032 \ifx\bbl@patterns@\relax
5033 \let\bbl@patterns@\empty
5034 \fi
5035 \ifx\bbl@pttnlist@\empty\else
5036 \bbl@warning{%
5037 You must not intermingle \string\selectlanguage\space and\%
5038 \string\babelpatterns\space or some patterns will not\%
5039 be taken into account. Reported}%
5040 \fi
5041 \ifx\@empty#1%
5042 \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5043 \else
5044 \edef\bbl@tempb{\zap@space#1 \empty}%
5045 \bbl@for\bbl@tempa\bbl@tempb{%
5046 \bbl@fixname\bbl@tempa
5047 \bbl@iflanguage\bbl@tempa{%
5048 \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5049 \@ifundefined{bbl@patterns@\bbl@tempa}%
5050 \empty
5051 {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5052 #2}}}%
5053 \fi}}

```

### 13.4 Southeast Asian scripts

First, some general code for line breaking, used by `\babelposthyphenation`. 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. See Unicode UAX 14.

```

5054% TODO - to a lua file
5055 \directlua{
5056 Babel = Babel or {}

```

```

5057 Babel.linebreaking = Babel.linebreaking or {}
5058 Babel.linebreaking.before = {}
5059 Babel.linebreaking.after = {}
5060 Babel.locale = {} % Free to use, indexed with \localeid
5061 function Babel.linebreaking.add_before(func)
5062     tex.print([[ \noexpand\csname bbl@luahyphenate\endcsname ]])
5063     table.insert(Babel.linebreaking.before, func)
5064 end
5065 function Babel.linebreaking.add_after(func)
5066     tex.print([[ \noexpand\csname bbl@luahyphenate\endcsname ]])
5067     table.insert(Babel.linebreaking.after, func)
5068 end
5069 }
5070 \def\bbl@intraspace#1 #2 #3\@{
5071     \directlua{
5072         Babel = Babel or {}
5073         Babel.intraspaces = Babel.intraspaces or {}
5074         Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5075             {b = #1, p = #2, m = #3}
5076         Babel.locale_props[\the\localeid].intraspace = %
5077             {b = #1, p = #2, m = #3}
5078     }}
5079 \def\bbl@intrapenalty#1\@{
5080     \directlua{
5081         Babel = Babel or {}
5082         Babel.intrapenalties = Babel.intrapenalties or {}
5083         Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5084         Babel.locale_props[\the\localeid].intrapenalty = #1
5085     }}
5086 \begingroup
5087 \catcode`\%=12
5088 \catcode`\^=14
5089 \catcode`\'=12
5090 \catcode`\~=12
5091 \gdef\bbl@seaintraspace{^
5092     \let\bbl@seaintraspace\relax
5093     \directlua{
5094         Babel = Babel or {}
5095         Babel.sea_enabled = true
5096         Babel.sea_ranges = Babel.sea_ranges or {}
5097         function Babel.set_chranges (script, chrng)
5098             local c = 0
5099             for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5100                 Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5101                 c = c + 1
5102             end
5103         end
5104         function Babel.sea_disc_to_space (head)
5105             local sea_ranges = Babel.sea_ranges
5106             local last_char = nil
5107             local quad = 655360 ^% 10 pt = 655360 = 10 * 65536
5108             for item in node.traverse(head) do
5109                 local i = item.id
5110                 if i == node.id'glyph' then
5111                     last_char = item
5112                 elseif i == 7 and item.subtype == 3 and last_char
5113                     and last_char.char > 0x0C99 then
5114                     quad = font.getfont(last_char.font).size
5115                     for lg, rg in pairs(sea_ranges) do

```

```

5116         if last_char.char > rg[1] and last_char.char < rg[2] then
5117             lg = lg:sub(1, 4) ^% Remove trailing number of, eg, Cyril
5118             local intraspace = Babel.intraspaces[lg]
5119             local intrapenalty = Babel.intrapenalties[lg]
5120             local n
5121             if intrapenalty ~= 0 then
5122                 n = node.new(14, 0) ^% penalty
5123                 n.penalty = intrapenalty
5124                 node.insert_before(head, item, n)
5125             end
5126             n = node.new(12, 13) ^% (glue, spaceskip)
5127             node.setglue(n, intraspace.b * quad,
5128                          intraspace.p * quad,
5129                          intraspace.m * quad)
5130             node.insert_before(head, item, n)
5131             node.remove(head, item)
5132         end
5133     end
5134 end
5135 end
5136 end
5137 }^^
5138 \bbl@luahyphenate}
5139 \catcode`\%=14
5140 \gdef\bbl@cjkintraspaces{%
5141 \let\bbl@cjkintraspaces\relax
5142 \directlua{
5143     Babel = Babel or {}
5144     require'babel-data-cjk.lua'
5145     Babel.cjk_enabled = true
5146     function Babel.cjk_linebreak(head)
5147         local GLYPH = node.id'glyph'
5148         local last_char = nil
5149         local quad = 655360 % 10 pt = 655360 = 10 * 65536
5150         local last_class = nil
5151         local last_lang = nil
5152
5153         for item in node.traverse(head) do
5154             if item.id == GLYPH then
5155
5156                 local lang = item.lang
5157
5158                 local LOCALE = node.get_attribute(item,
5159             luatexbase.registernumber'bbl@attr@locale')
5160                 local props = Babel.locale_props[LOCALE]
5161
5162                 local class = Babel.cjk_class[item.char].c
5163
5164                 if class == 'cp' then class = 'cl' end % )] as CL
5165                 if class == 'id' then class = 'I' end
5166
5167                 local br = 0
5168                 if class and last_class and Babel.cjk_breaks[last_class][class] then
5169                     br = Babel.cjk_breaks[last_class][class]
5170                 end
5171
5172                 if br == 1 and props.linebreak == 'c' and
5173                     lang ~= \the\l@nohyphenation\space and
5174                     last_lang ~= \the\l@nohyphenation then

```



```

5175         local intrapenalty = props.intrapenalty
5176         if intrapenalty ~= 0 then
5177             local n = node.new(14, 0)    % penalty
5178             n.penalty = intrapenalty
5179             node.insert_before(head, item, n)
5180         end
5181         local intraspace = props.intraspace
5182         local n = node.new(12, 13)    % (glue, spaceskip)
5183         node.setglue(n, intraspace.b * quad,
5184                     intraspace.p * quad,
5185                     intraspace.m * quad)
5186         node.insert_before(head, item, n)
5187     end
5188
5189     if font.getfont(item.font) then
5190         quad = font.getfont(item.font).size
5191     end
5192     last_class = class
5193     last_lang = lang
5194     else % if penalty, glue or anything else
5195         last_class = nil
5196     end
5197 end
5198 lang.hyphenate(head)
5199 end
5200 }%
5201 \bbl@luahyphenate}
5202 \gdef\bbl@luahyphenate{%
5203 \let\bbl@luahyphenate\relax
5204 \directlua{
5205     luatexbase.add_to_callback('hyphenate',
5206     function (head, tail)
5207         if Babel.linebreaking.before then
5208             for k, func in ipairs(Babel.linebreaking.before) do
5209                 func(head)
5210             end
5211         end
5212         if Babel.cjk_enabled then
5213             Babel.cjk_linebreak(head)
5214         end
5215         lang.hyphenate(head)
5216         if Babel.linebreaking.after then
5217             for k, func in ipairs(Babel.linebreaking.after) do
5218                 func(head)
5219             end
5220         end
5221         if Babel.sea_enabled then
5222             Babel.sea_disc_to_space(head)
5223         end
5224     end,
5225     'Babel.hyphenate')
5226 }
5227 }
5228 \endgroup
5229 \def\bbl@provide@intraspace{%
5230 \bbl@ifunset{\bbl@intsp@language}{%
5231     {\expandafter\ifx\curname\bbl@intsp@language\endcsname\@empty\else
5232         \bbl@xin@{\bbl@cl{lnbrk}}{c}%
5233         \fin@           % cjk

```

```

5234      \bbl@cjkintraspacespace
5235      \directlua{
5236          Babel = Babel or {}
5237          Babel.locale_props = Babel.locale_props or {}
5238          Babel.locale_props[\the\localeid].linebreak = 'c'
5239      }%
5240      \bbl@exp{\bbl@intraspacespace\bbl@cl{intsp}\bbl@cl{intsp}}\bbl@cl{intsp}%
5241      \ifx\bbl@KVP@intrapenalty\@nil
5242          \bbl@intrapenalty0\@
5243      \fi
5244      \else          % sea
5245          \bbl@seaintraspacespace
5246          \bbl@exp{\bbl@intraspacespace\bbl@cl{intsp}\bbl@cl{intsp}}\bbl@cl{intsp}%
5247          \directlua{
5248              Babel = Babel or {}
5249              Babel.sea_ranges = Babel.sea_ranges or {}
5250              Babel.set_chranges('\bbl@cl{sbcpr}',
5251                              '\bbl@cl{chrng}')
5252          }%
5253          \ifx\bbl@KVP@intrapenalty\@nil
5254              \bbl@intrapenalty0\@
5255          \fi
5256      \fi
5257      \fi
5258      \ifx\bbl@KVP@intrapenalty\@nil\else
5259          \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
5260      \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.

```

5261 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5262 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfont}
5263 \DisableBabelHook{babel-fontspec}
5264 <<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.

```

5265% TODO - to a lua file
5266 \directlua{
5267 Babel.script_blocks = {
5268   ['dflt'] = {},
5269   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5270              {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},

```

```

5271 ['Armn'] = {{0x0530, 0x058F}},
5272 ['Beng'] = {{0x0980, 0x09FF}},
5273 ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5274 ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5275 ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5276             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5277 ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5278 ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5279             {0xAB00, 0xAB2F}},
5280 ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5281 % Don't follow strictly Unicode, which places some Coptic letters in
5282 % the 'Greek and Coptic' block
5283 ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5284 ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5285             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5286             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5287             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5288             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5289             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5290 ['Hebr'] = {{0x0590, 0x05FF}},
5291 ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5292             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5293 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5294 ['Knda'] = {{0x0C80, 0x0CFF}},
5295 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5296             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5297             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5298 ['Lao0'] = {{0x0E80, 0x0EFF}},
5299 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5300             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5301             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5302 ['Mahj'] = {{0x11150, 0x1117F}},
5303 ['Mlym'] = {{0x0D00, 0x0D7F}},
5304 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5305 ['Orya'] = {{0x0B00, 0x0B7F}},
5306 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5307 ['Syr1'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5308 ['Taml'] = {{0x0B80, 0x0BFF}},
5309 ['Telu'] = {{0x0C00, 0x0C7F}},
5310 ['Tfng'] = {{0x2D30, 0x2D7F}},
5311 ['Thai'] = {{0x0E00, 0x0E7F}},
5312 ['Tibt'] = {{0x0F00, 0x0FFF}},
5313 ['Vaii'] = {{0xA500, 0xA63F}},
5314 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5315 }
5316
5317 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyr1
5318 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5319 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5320
5321 function Babel.locale_map(head)
5322   if not Babel.locale_mapped then return head end
5323
5324   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
5325   local GLYPH = node.id('glyph')
5326   local inmath = false
5327   local toloc_save
5328   for item in node.traverse(head) do
5329     local toloc

```

```

5330   if not inmath and item.id == GLYPH then
5331     % Optimization: build a table with the chars found
5332     if Babel.chr_to_loc[item.char] then
5333       toloc = Babel.chr_to_loc[item.char]
5334     else
5335       for lc, maps in pairs(Babel.loc_to_scr) do
5336         for _, rg in pairs(maps) do
5337           if item.char >= rg[1] and item.char <= rg[2] then
5338             Babel.chr_to_loc[item.char] = lc
5339             toloc = lc
5340             break
5341           end
5342         end
5343       end
5344     end
5345     % Now, take action, but treat composite chars in a different
5346     % fashion, because they 'inherit' the previous locale. Not yet
5347     % optimized.
5348     if not toloc and
5349       (item.char >= 0x0300 and item.char <= 0x036F) or
5350       (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5351       (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5352       toloc = toloc_save
5353     end
5354     if toloc and toloc > -1 then
5355       if Babel.locale_props[toloc].lg then
5356         item.lang = Babel.locale_props[toloc].lg
5357         node.set_attribute(item, LOCALE, toloc)
5358       end
5359       if Babel.locale_props[toloc]['/'..item.font] then
5360         item.font = Babel.locale_props[toloc]['/'..item.font]
5361       end
5362       toloc_save = toloc
5363     end
5364     elseif not inmath and item.id == 7 then
5365       item.replace = item.replace and Babel.locale_map(item.replace)
5366       item.pre      = item.pre and Babel.locale_map(item.pre)
5367       item.post     = item.post and Babel.locale_map(item.post)
5368     elseif item.id == node.id'math' then
5369       inmath = (item.subtype == 0)
5370     end
5371   end
5372   return head
5373 end
5374 }

```

The code for \babelcharproperty is straightforward. Just note the modified lua table can be different.

```

5375 \newcommand\babelcharproperty[1]{%
5376   \count@=#1\relax
5377   \ifvmode
5378     \expandafter\bbl@chprop
5379   \else
5380     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5381               vertical mode (preamble or between paragraphs)}%
5382     {See the manual for futher info}%
5383   \fi}
5384 \newcommand\bbl@chprop[3][\the\count@]{%
5385   \@tempcnta=#1\relax

```

```

5386 \bbl@ifunset{bbl@chprop@#2}%
5387 {\bbl@error{No property named '#2'. Allowed values are\\%
5388             direction (bc), mirror (bmg), and linebreak (lb)}}%
5389             {See the manual for futher info}}%
5390 {}%
5391 \loop
5392 \bbl@cs{chprop@#2}{#3}%
5393 \ifnum\count@<\@tempcnta
5394 \advance\count@\@ne
5395 \repeat}
5396 \def\bbl@chprop@direction#1{%
5397 \directlua{
5398   Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5399   Babel.characters[\the\count@]['d'] = '#1'
5400 }}
5401 \let\bbl@chprop@bc\bbl@chprop@direction
5402 \def\bbl@chprop@mirror#1{%
5403 \directlua{
5404   Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5405   Babel.characters[\the\count@]['m'] = '\number#1'
5406 }}
5407 \let\bbl@chprop@bmg\bbl@chprop@mirror
5408 \def\bbl@chprop@linebreak#1{%
5409 \directlua{
5410   Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5411   Babel.cjk_characters[\the\count@]['c'] = '#1'
5412 }}
5413 \let\bbl@chprop@lb\bbl@chprop@linebreak
5414 \def\bbl@chprop@locale#1{%
5415 \directlua{
5416   Babel.chr_to_loc = Babel.chr_to_loc or {}
5417   Babel.chr_to_loc[\the\count@] =
5418     \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
5419 }}

```

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.

```

5420 \begingroup % TODO - to a lua file
5421 \catcode`\~ = 12
5422 \catcode`\# = 12
5423 \catcode`\% = 12
5424 \catcode`\& = 14
5425 \directlua{
5426   Babel.linebreaking.replacements = {}
5427   Babel.linebreaking.replacements[0] = {} &% pre
5428   Babel.linebreaking.replacements[1] = {} &% post
5429
5430   &% Discretionaries contain strings as nodes

```

```

5431 function Babel.str_to_nodes(fn, matches, base)
5432     local n, head, last
5433     if fn == nil then return nil end
5434     for s in string.utfvalues(fn(matches)) do
5435         if base.id == 7 then
5436             base = base.replace
5437         end
5438         n = node.copy(base)
5439         n.char = s
5440         if not head then
5441             head = n
5442         else
5443             last.next = n
5444         end
5445         last = n
5446     end
5447     return head
5448 end
5449
5450 Babel.fetch_subtext = {}
5451
5452 %% Merging both functions doesn't seem feasible, because there are too
5453 %% many differences.
5454 Babel.fetch_subtext[0] = function(head)
5455     local word_string = ''
5456     local word_nodes = {}
5457     local lang
5458     local item = head
5459     local inmath = false
5460
5461     while item do
5462         if item.id == 11 then
5463             inmath = (item.subtype == 0)
5464         end
5465
5466         if inmath then
5467             %% pass
5468         end
5469
5470         elseif item.id == 29 then
5471             local locale = node.get_attribute(item, Babel.attr_locale)
5472
5473             if lang == locale or lang == nil then
5474                 if (item.char ~= 124) then %% ie, not | = space
5475                     lang = lang or locale
5476                     word_string = word_string .. unicode.utf8.char(item.char)
5477                     word_nodes[#word_nodes+1] = item
5478                 end
5479             else
5480                 break
5481             end
5482
5483             elseif item.id == 12 and item.subtype == 13 then
5484                 word_string = word_string .. '|'
5485                 word_nodes[#word_nodes+1] = item
5486
5487             %% Ignore leading unrecognized nodes, too.
5488             elseif word_string ~= '' then
5489                 word_string = word_string .. Babel.us_char

```

```

5490         word_nodes[#word_nodes+1] = item  %% Will be ignored
5491     end
5492
5493     item = item.next
5494 end
5495
5496     %% Here and above we remove some trailing chars but not the
5497     %% corresponding nodes. But they aren't accessed.
5498     if word_string:sub(-1) == '|' then
5499         word_string = word_string:sub(1,-2)
5500     end
5501     word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
5502     return word_string, word_nodes, item, lang
5503 end
5504
5505 Babel.fetch_subtext[1] = function(head)
5506     local word_string = ''
5507     local word_nodes = {}
5508     local lang
5509     local item = head
5510     local inmath = false
5511
5512     while item do
5513
5514         if item.id == 11 then
5515             inmath = (item.subtype == 0)
5516         end
5517
5518         if inmath then
5519             %% pass
5520
5521         elseif item.id == 29 then
5522             if item.lang == lang or lang == nil then
5523                 if (item.char ~= 124) and (item.char ~= 61) then %% not =, not |
5524                     lang = lang or item.lang
5525                     word_string = word_string .. unicode.utf8.char(item.char)
5526                     word_nodes[#word_nodes+1] = item
5527                 end
5528             else
5529                 break
5530             end
5531
5532         elseif item.id == 7 and item.subtype == 2 then
5533             word_string = word_string .. '='
5534             word_nodes[#word_nodes+1] = item
5535
5536         elseif item.id == 7 and item.subtype == 3 then
5537             word_string = word_string .. '|'
5538             word_nodes[#word_nodes+1] = item
5539
5540             %% (1) Go to next word if nothing was found, and (2) implicitly
5541             %% remove leading USs.
5542             elseif word_string == '' then
5543                 %% pass
5544
5545             %% This is the responsible for splitting by words.
5546             elseif (item.id == 12 and item.subtype == 13) then
5547                 break
5548

```

```

5549     else
5550         word_string = word_string .. Babel.us_char
5551         word_nodes[#word_nodes+1] = item  %% Will be ignored
5552     end
5553
5554     item = item.next
5555 end
5556
5557 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
5558 return word_string, word_nodes, item, lang
5559 end
5560
5561 function Babel.pre_hyphenate_replace(head)
5562     Babel.hyphenate_replace(head, 0)
5563 end
5564
5565 function Babel.post_hyphenate_replace(head)
5566     Babel.hyphenate_replace(head, 1)
5567 end
5568
5569 Babel.us_char = string.char(31)
5570
5571 function Babel.hyphenate_replace(head, mode)
5572     local u = unicode.utf8
5573     local lbkr = Babel.linebreaking.replacements[mode]
5574
5575     local word_head = head
5576
5577     while true do  %% for each subtext block
5578
5579         local w, wn, nw, lang = Babel.fetch_subtext[mode](word_head)
5580
5581         if Babel.debug then
5582             print()
5583             print('@@@@', w, nw)
5584         end
5585
5586         if nw == nil and w == '' then break end
5587
5588         if not lang then goto next end
5589         if not lbkr[lang] then goto next end
5590
5591         %% For each saved (pre|post)hyphenation. TODO. Reconsider how
5592         %% loops are nested.
5593         for k=1, #lbkr[lang] do
5594             local p = lbkr[lang][k].pattern
5595             local r = lbkr[lang][k].replace
5596
5597             if Babel.debug then
5598                 print('====', p, mode)
5599             end
5600
5601             %% This variable is set in some cases below to the first *byte*
5602             %% after the match, either as found by u.match (faster) or the
5603             %% computed position based on sc if w has changed.
5604             local last_match = 0
5605
5606             %% For every match.
5607             while true do

```



```

5608         if Babel.debug then
5609             print('-----')
5610         end
5611         local new  %% used when inserting and removing nodes
5612         local refetch = false
5613
5614         local matches = { u.match(w, p, last_match) }
5615         if #matches < 2 then break end
5616
5617         %% Get and remove empty captures (with ())'s, which return a
5618         %% number with the position), and keep actual captures
5619         %% (from (...)), if any, in matches.
5620         local first = table.remove(matches, 1)
5621         local last  = table.remove(matches, #matches)
5622         %% Non re-fetched substrings may contain \31, which separates
5623         %% subsubstrings.
5624         if string.find(w:sub(first, last-1), Babel.us_char) then break end
5625
5626         local save_last = last %% with A()BC()D, points to D
5627
5628         %% Fix offsets, from bytes to unicode. Explained above.
5629         first = u.len(w:sub(1, first-1)) + 1
5630         last  = u.len(w:sub(1, last-1)) %% now last points to C
5631
5632         if Babel.debug then
5633             print(p)
5634             print('', 'sc', 'first', 'last', 'last_m', 'w')
5635         end
5636
5637         %% This loop traverses the matched substring and takes the
5638         %% corresponding action stored in the replacement list.
5639         %% sc = the position in substr nodes / string
5640         %% rc = the replacement table index
5641         local sc = first-1
5642         local rc = 0
5643         while rc < last-first+1 do %% for each replacement
5644             if Babel.debug then
5645                 print('.....')
5646             end
5647             sc = sc + 1
5648             rc = rc + 1
5649             local crep = r[rc]
5650             local char_node = wn[sc]
5651             local char_base = char_node
5652             local end_replacement = false
5653
5654             if crep and crep.data then
5655                 char_base = wn[crep.data+first-1]
5656             end
5657
5658             if Babel.debug then
5659                 print('*', sc, first, last, last_match, w)
5660             end
5661
5662             if crep and next(crep) == nil then %% {}
5663                 last_match = save_last
5664
5665             elseif crep == nil then %% remove
5666                 node.remove(head, char_node)

```

```

5667         table.remove(wn, sc)
5668         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
5669         last_match = utf8.offset(w, sc)
5670         sc = sc - 1  && Nothing has been inserted
5671
5672     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
5673         local d = node.new(7, 0)  && (disc, discretionary)
5674         d.pre    = Babel.str_to_nodes(crep.pre, matches, char_base)
5675         d.post    = Babel.str_to_nodes(crep.post, matches, char_base)
5676         d.replace = Babel.str_to_nodes(crep.no, matches, char_base)
5677         d.attr = char_base.attr
5678         if crep.pre == nil then  && TeXbook p96
5679             d.penalty = crep.penalty or tex.hyphenpenalty
5680         else
5681             d.penalty = crep.penalty or tex.exhyphenpenalty
5682         end
5683         head, new = node.insert_before(head, char_node, d)
5684         end_replacement = true
5685
5686     elseif crep and crep.penalty then
5687         local d = node.new(14, 0)  && (penalty, userpenalty)
5688         d.attr = char_base.attr
5689         d.penalty = crep.penalty
5690         head, new = node.insert_before(head, char_node, d)
5691         end_replacement = true
5692
5693     elseif crep and crep.string then
5694         local str = crep.string(matches)
5695         if str == '' then  && Gather with nil
5696             refetch = true
5697             if sc == 1 then
5698                 word_head = char_node.next
5699             end
5700             head, new = node.remove(head, char_node)
5701         elseif char_node.id == 29 and u.len(str) == 1 then
5702             char_node.char = string.utfvalue(str)
5703             w = u.sub(w, 1, sc-1) .. str .. u.sub(w, sc+1)
5704             last_match = utf8.offset(w, sc+1)
5705         else
5706             refetch = true
5707             local n
5708             for s in string.utfvalues(str) do
5709                 if char_node.id == 7 then
5710                     && TODO. Remove this limitation.
5711                     texio.write_nl('Automatic hyphens cannot be replaced, just removed.')
5712                 else
5713                     n = node.copy(char_base)
5714                 end
5715                 n.char = s
5716                 if sc == 1 then
5717                     head, new = node.insert_before(head, char_node, n)
5718                     word_head = new
5719                 else
5720                     node.insert_before(head, char_node, n)
5721                 end
5722             end
5723             node.remove(head, char_node)
5724         end  && string length
5725     end  && if char and char.string (ie replacement cases)

```

```

5726
5727         %% Shared by disc and penalty.
5728         if end_replacement then
5729             if sc == 1 then
5730                 word_head = new
5731             end
5732             if crep.insert then
5733                 last_match = save_last
5734             else
5735                 node.remove(head, char_node)
5736                 w = u.sub(w, 1, sc-1) .. Babel.us_char .. u.sub(w, sc+1)
5737                 last_match = utf8.offset(w, sc)
5738             end
5739         end
5740     end %% for each replacement
5741
5742     if Babel.debug then
5743         print('/', sc, first, last, last_match, w)
5744     end
5745
5746     %% TODO. refetch will be eventually unnecessary.
5747     if refetch then
5748         w, wn, nw, lang = Babel.fetch_subtext[mode](word_head)
5749     end
5750
5751     end %% for match
5752 end %% for patterns
5753
5754 ::next::
5755     word_head = nw
5756 end %% for substring
5757 return head
5758 end
5759
5760 %% This table stores capture maps, numbered consecutively
5761 Babel.capture_maps = {}
5762
5763 %% The following functions belong to the next macro
5764 function Babel.capture_func(key, cap)
5765     local ret = "[" .. cap:gsub('{{[0-9]}}', "[]..m[%1]..[(") .. "]"
5766     ret = ret:gsub('{{[0-9]}|([^\]|+)|(.-)}', Babel.capture_func_map)
5767     ret = ret:gsub("%[%]%%.%", '')
5768     ret = ret:gsub("%.%[%]%%", '')
5769     return key .. "[=function(m) return ] .. ret .. [ end]"
5770 end
5771
5772 function Babel.capt_map(from, mapno)
5773     return Babel.capture_maps[mapno][from] or from
5774 end
5775
5776 %% Handle the {n|abc|ABC} syntax in captures
5777 function Babel.capture_func_map(capno, from, to)
5778     local froms = {}
5779     for s in string.utfcharacters(from) do
5780         table.insert(froms, s)
5781     end
5782     local cnt = 1
5783     table.insert(Babel.capture_maps, {})
5784     local mlen = table.getn(Babel.capture_maps)

```

```

5785   for s in string.utfcharacters(to) do
5786       Babel.capture_maps[mlen][from[cnt]] = s
5787       cnt = cnt + 1
5788   end
5789   return "]]..Babel.capt_map(m[" .. capno .. "], " ..
5790       (mlen) .. ").." .. "["
5791 end
5792 }

```

Now the  $\TeX$  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  $\text{function}(m)$   $\text{return } m[1]..m[1]..'-'$  end, where  $m$  are the matches returned after applying the pattern. With a mapped capture the functions are similar to  $\text{function}(m)$   $\text{return } \text{Babel.capt\_map}(m[1],1)$  end, where the last argument identifies the mapping to be applied to  $m[1]$ . The way it is carried out is somewhat tricky, but the effect is not dissimilar to `lua load - save the code as string in a TeX macro, and expand this macro at the appropriate place`. As `\directlua` does not take into account the current catcode of `@`, we just avoid this character in macro names (which explains the internal group, too).

```

5793 \catcode\#=#6
5794 \gdef\babelposthyphenation#1#2#3{&%
5795   \bbl@activateposthyphen
5796   \begingroup
5797     \def\babeltempa{\bbl@add@list\babeltempb}&%
5798     \let\babeltempb\@empty
5799     \bbl@foreach{#3}{&%
5800       \bbl@ifsamestring{##1}{remove}&%
5801       {\bbl@add@list\babeltempb{nil}}&%
5802       {\directlua{
5803         local rep = {[##1]}
5804         rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5805         rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
5806         rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5807         rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
5808         rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5809         tex.print([[string\babeltempa{}} .. rep .. [{}]])
5810       }}&%
5811     \directlua{
5812       local lbkr = Babel.linebreaking.replacements[1]
5813       local u = unicode.utf8
5814       &% Convert pattern:
5815       local patt = string.gsub([=[#2]=], '%s', '')
5816       if not u.find(patt, '()', nil, true) then
5817         patt = '()' .. patt .. '()'
5818       end
5819       patt = string.gsub(patt, '%(%)%', '^()')
5820       patt = string.gsub(patt, '%$(%)%', '()$')
5821       patt = u.gsub(patt, '{(.)}',
5822         function (n)
5823           return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5824         end)
5825       lbkr[\the\csname l@#1\endcsname] = lbkr[\the\csname l@#1\endcsname] or {}
5826       table.insert(lbkr[\the\csname l@#1\endcsname],
5827         { pattern = patt, replace = { \babeltempb } })
5828     }&%
5829   \endgroup}
5830 % TODO. Copy paste pattern.
5831 \gdef\babelprehyphenation#1#2#3{&%
5832   \bbl@activateprehyphen
5833   \begingroup

```

```

5834 \def\babeltempa{\bbl@add@list\babeltempb}&%
5835 \let\babeltempb\@empty
5836 \bbl@foreach{#3}{&%
5837   \bbl@ifsamestring{##1}{remove}&%
5838   {\bbl@add@list\babeltempb{nil}}&%
5839   {\directlua{
5840     local rep = {[##1]}
5841     rep = rep:gsub('^%s*(insert)%s*', ' ', 'insert = true, ')
5842     rep = rep:gsub('(string)%s*=%s*([%s,]*)', Babel.capture_func)
5843     tex.print([[\\string\babeltempa{}}] .. rep .. [[]]{}]}
5844   }}&%
5845 \directlua{
5846   local lbkr = Babel.linebreaking.replacements[0]
5847   local u = unicode.utf8
5848   &% Convert pattern:
5849   local patt = string.gsub(#[2]=], '%s', ' ')
5850   if not u.find(patt, '()', nil, true) then
5851     patt = '()' .. patt .. '()'
5852   end
5853   &% patt = string.gsub(patt, '%(%)%', '^()')
5854   &% patt = string.gsub(patt, '([%{}])%$%$', '%1()$')
5855   patt = u.gsub(patt, '{(.)}',
5856     function (n)
5857       return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5858     end)
5859   lbkr[\the\csname bbl@id@@#1\endcsname] = lbkr[\the\csname bbl@id@@#1\endcsname] or {}
5860   table.insert(lbkr[\the\csname bbl@id@@#1\endcsname],
5861     { pattern = patt, replace = { \babeltempb } })
5862   }&%
5863 \endgroup}
5864 \endgroup
5865 \def\bbl@activateposthyphen{%
5866   \let\bbl@activateposthyphen\relax
5867   \directlua{
5868     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5869   }}
5870 \def\bbl@activateprehyphen{%
5871   \let\bbl@activateprehyphen\relax
5872   \directlua{
5873     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5874   }}

```

## 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`, `hhline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```

5875 \bbl@trace{Redefinitions for bidi layout}
5876 \ifx\@eqnnum\@undefined\else

```

```

5877 \ifx\bb1@attr@dir\@undefined\else
5878 \edef\@eqnnum{%
5879 \unexpanded{\ifcase\bb1@attr@dir\else\bb1@textdir\@ne\fi}%
5880 \unexpanded\expandafter{\@eqnnum}}
5881 \fi
5882 \fi
5883 \ifx\bb1@opt@layout\@nnil\endinput\fi % if no layout
5884 \ifnum\bb1@bidimode>\z@
5885 \def\bb1@nextfake#1{% non-local changes, use always inside a group!
5886 \bb1@exp{%
5887 \mathdir\the\bodydir
5888 #1% Once entered in math, set boxes to restore values
5889 \<ifmmode>%
5890 \everyvbox{%
5891 \the\everyvbox
5892 \bodydir\the\bodydir
5893 \mathdir\the\mathdir
5894 \everyhbox{\the\everyhbox}%
5895 \everyvbox{\the\everyvbox}}%
5896 \everyhbox{%
5897 \the\everyhbox
5898 \bodydir\the\bodydir
5899 \mathdir\the\mathdir
5900 \everyhbox{\the\everyhbox}%
5901 \everyvbox{\the\everyvbox}}%
5902 \<fi>}}%
5903 \def\@hangfrom#1{%
5904 \setbox\@tempboxa\hbox{{#1}}%
5905 \hangindent\wd\@tempboxa
5906 \ifnum\bb1@getluadir{page}=\bb1@getluadir{par}\else
5907 \shapemode\@ne
5908 \fi
5909 \noindent\box\@tempboxa}
5910 \fi
5911 \IfBabelLayout{tabular}
5912 {\let\bb1@OL@tabular\@tabular
5913 \bb1@replace\@tabular{$}\{\bb1@nextfake$}%
5914 \let\bb1@NL@tabular\@tabular
5915 \AtBeginDocument{%
5916 \ifx\bb1@NL@tabular\@tabular\else
5917 \bb1@replace\@tabular{$}\{\bb1@nextfake$}%
5918 \let\bb1@NL@tabular\@tabular
5919 \fi}}
5920 {}
5921 \IfBabelLayout{lists}
5922 {\let\bb1@OL@list\list
5923 \bb1@sreplace\list{\parshape}\{\bb1@listparshape}%
5924 \let\bb1@NL@list\list
5925 \def\bb1@listparshape#1#2#3{%
5926 \parshape #1 #2 #3 %
5927 \ifnum\bb1@getluadir{page}=\bb1@getluadir{par}\else
5928 \shapemode\tw@
5929 \fi}}
5930 {}
5931 \IfBabelLayout{graphics}
5932 {\let\bb1@pictresetdir\relax
5933 \def\bb1@pictsetdir#1{%
5934 \ifcase\bb1@thetextdir
5935 \let\bb1@pictresetdir\relax

```

```

5936 \else
5937 \bodydir TLT
5938 % \(\text|par)dir required in pgf:
5939 \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
5940 \fi}%
5941 \ifx\AddToHook\undefined\else
5942 \AddToHook{env/picture/begin}{\bbl@pictsetdir\z@}%
5943 \fi
5944 \AtBeginDocument
5945 {\ifx\tikz@atbegin@node\undefined\else
5946 \let\bbl@OL@pgfpicture\pgfpicture
5947 \bbl@sreplace\pgfpicture{\pgfpicturetrue}%
5948 {\bbl@pictsetdir\@ne\pgfpicturetrue}%
5949 \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\@ne}%
5950 \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
5951 \fi}}
5952 {}

```

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`.

```

5953 \IfBabelLayout{counters}%
5954 {\let\bbl@OL@@textsuperscript\textsuperscript
5955 \bbl@sreplace\textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
5956 \let\bbl@latinarabic=\@arabic
5957 \let\bbl@OL@@arabic\@arabic
5958 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5959 \@ifpackagewith{babel}{bidi=default}%
5960 {\let\bbl@asciroman=\@roman
5961 \let\bbl@OL@@roman\@roman
5962 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5963 \let\bbl@asciiRoman=\@Roman
5964 \let\bbl@OL@@roman\@Roman
5965 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
5966 \let\bbl@OL@labelenumii\labelenumii
5967 \def\labelenumii{}\theenumii{}%
5968 \let\bbl@OL@p@enumiii\p@enumiii
5969 \def\p@enumiii{\p@enumii}\theenumii{}\{\}\{\}}
5970 <<Footnote changes>>
5971 \IfBabelLayout{footnotes}%
5972 {\let\bbl@OL@footnote\footnote
5973 \BabelFootnote\footnote\languagename{}\{\}%
5974 \BabelFootnote\localfootnote\languagename{}\{\}%
5975 \BabelFootnote\mainfootnote{}\{\}\{\}}
5976 {}

```

Some  $\TeX$  macros use internally the math mode for text formatting. They have very little in common and are grouped here, as a single option.

```

5977 \IfBabelLayout{extras}%
5978 {\let\bbl@OL@underline\underline
5979 \bbl@sreplace\underline{\$@@underline}{\bbl@nextfake\$@@underline}%
5980 \let\bbl@OL@LaTeX2e\LaTeX2e
5981 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
5982 \if b\expandafter\@car\@series\@nil\boldmath\fi
5983 \babelsublr{%
5984 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}
5985 {}
5986 </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.

```
5987 (*basic-r)
5988 Babel = Babel or {}
5989
5990 Babel.bidi_enabled = true
5991
5992 require('babel-data-bidi.lua')
5993
5994 local characters = Babel.characters
5995 local ranges = Babel.ranges
5996
5997 local DIR = node.id("dir")
5998
5999 local function dir_mark(head, from, to, outer)
6000   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6001   local d = node.new(DIR)
6002   d.dir = '+' .. dir
6003   node.insert_before(head, from, d)
6004   d = node.new(DIR)
6005   d.dir = '-' .. dir
6006   node.insert_after(head, to, d)
6007 end
```



```

6008
6009 function Babel.bidi(head, ispar)
6010   local first_n, last_n          -- first and last char with nums
6011   local last_es                  -- an auxiliary 'last' used with nums
6012   local first_d, last_d          -- first and last char in L/R block
6013   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):

```

6014   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6015   local strong_lr = (strong == 'l') and 'l' or 'r'
6016   local outer = strong
6017
6018   local new_dir = false
6019   local first_dir = false
6020   local inmath = false
6021
6022   local last_lr
6023
6024   local type_n = ''
6025
6026   for item in node.traverse(head) do
6027
6028     -- three cases: glyph, dir, otherwise
6029     if item.id == node.id'glyph'
6030       or (item.id == 7 and item.subtype == 2) then
6031
6032       local itemchar
6033       if item.id == 7 and item.subtype == 2 then
6034         itemchar = item.replace.char
6035       else
6036         itemchar = item.char
6037       end
6038       local chardata = characters[itemchar]
6039       dir = chardata and chardata.d or nil
6040       if not dir then
6041         for nn, et in ipairs(ranges) do
6042           if itemchar < et[1] then
6043             break
6044           elseif itemchar <= et[2] then
6045             dir = et[3]
6046             break
6047           end
6048         end
6049       end
6050       dir = dir or 'l'
6051       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).

```

6052   if new_dir then
6053     attr_dir = 0
6054     for at in node.traverse(item.attr) do
6055       if at.number == luatexbase.registernumber'bbl@attr@dir' then
6056         attr_dir = at.value % 3

```

```

6057         end
6058     end
6059     if attr_dir == 1 then
6060         strong = 'r'
6061     elseif attr_dir == 2 then
6062         strong = 'al'
6063     else
6064         strong = 'l'
6065     end
6066     strong_lr = (strong == 'l') and 'l' or 'r'
6067     outer = strong_lr
6068     new_dir = false
6069 end
6070
6071 if dir == 'nsm' then dir = strong end          -- W1

```

**Numbers.** The dual <al>/<r> system for R is somewhat cumbersome.

```

6072     dir_real = dir          -- We need dir_real to set strong below
6073     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:

```

6074     if strong == 'al' then
6075         if dir == 'en' then dir = 'an' end          -- W2
6076         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6077         strong_lr = 'r'                                -- W3
6078     end

```

Once finished the basic setup for glyphs, consider the two other cases: dir node and the rest.

```

6079     elseif item.id == node.id'dir' and not inmath then
6080         new_dir = true
6081         dir = nil
6082     elseif item.id == node.id'math' then
6083         inmath = (item.subtype == 0)
6084     else
6085         dir = nil          -- Not a char
6086     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>.

```

6087     if dir == 'en' or dir == 'an' or dir == 'et' then
6088         if dir ~= 'et' then
6089             type_n = dir
6090         end
6091         first_n = first_n or item
6092         last_n = last_es or item
6093         last_es = nil
6094     elseif dir == 'es' and last_n then -- W3+W6
6095         last_es = item
6096     elseif dir == 'cs' then          -- it's right - do nothing
6097     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6098         if strong_lr == 'r' and type_n ~= '' then
6099             dir_mark(head, first_n, last_n, 'r')
6100         elseif strong_lr == 'l' and first_d and type_n == 'an' then
6101             dir_mark(head, first_n, last_n, 'r')
6102             dir_mark(head, first_d, last_d, outer)

```

```

6103     first_d, last_d = nil, nil
6104     elseif strong_lr == 'l' and type_n ~= '' then
6105         last_d = last_n
6106     end
6107     type_n = ''
6108     first_n, last_n = nil, nil
6109 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:

```

6110     if dir == 'l' or dir == 'r' then
6111         if dir ~= outer then
6112             first_d = first_d or item
6113             last_d = item
6114         elseif first_d and dir ~= strong_lr then
6115             dir_mark(head, first_d, last_d, outer)
6116             first_d, last_d = nil, nil
6117         end
6118     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>, resptly, 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.

```

6119     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6120         item.char = characters[item.char] and
6121             characters[item.char].m or item.char
6122     elseif (dir or new_dir) and last_lr ~= item then
6123         local mir = outer .. strong_lr .. (dir or outer)
6124         if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6125             for ch in node.traverse(node.next(last_lr)) do
6126                 if ch == item then break end
6127                 if ch.id == node.id'glyph' and characters[ch.char] then
6128                     ch.char = characters[ch.char].m or ch.char
6129                 end
6130             end
6131         end
6132     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).

```

6133     if dir == 'l' or dir == 'r' then
6134         last_lr = item
6135         strong = dir_real -- Don't search back - best save now
6136         strong_lr = (strong == 'l') and 'l' or 'r'
6137     elseif new_dir then
6138         last_lr = nil
6139     end
6140 end

```

Mirror the last chars if they are no directed. And make sure any open block is closed, too.

```

6141     if last_lr and outer == 'r' then
6142         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6143             if characters[ch.char] then
6144                 ch.char = characters[ch.char].m or ch.char
6145             end

```

```

6146     end
6147 end
6148 if first_n then
6149     dir_mark(head, first_n, last_n, outer)
6150 end
6151 if first_d then
6152     dir_mark(head, first_d, last_d, outer)
6153 end

```

In boxes, the dir node could be added before the original head, so the actual head is the previous node.

```

6154 return node.prev(head) or head
6155 end
6156 </basic-r>

```

And here the Lua code for bidi=basic:

```

6157 <(*basic>
6158 Babel = Babel or {}
6159
6160 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6161
6162 Babel.fontmap = Babel.fontmap or {}
6163 Babel.fontmap[0] = {}      -- l
6164 Babel.fontmap[1] = {}      -- r
6165 Babel.fontmap[2] = {}      -- al/an
6166
6167 Babel.bidi_enabled = true
6168 Babel.mirroring_enabled = true
6169
6170 require('babel-data-bidi.lua')
6171
6172 local characters = Babel.characters
6173 local ranges = Babel.ranges
6174
6175 local DIR = node.id('dir')
6176 local GLYPH = node.id('glyph')
6177
6178 local function insert_implicit(head, state, outer)
6179     local new_state = state
6180     if state.sim and state.eim and state.sim ~= state.eim then
6181         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6182         local d = node.new(DIR)
6183         d.dir = '+' .. dir
6184         node.insert_before(head, state.sim, d)
6185         local d = node.new(DIR)
6186         d.dir = '-' .. dir
6187         node.insert_after(head, state.eim, d)
6188     end
6189     new_state.sim, new_state.eim = nil, nil
6190     return head, new_state
6191 end
6192
6193 local function insert_numeric(head, state)
6194     local new
6195     local new_state = state
6196     if state.san and state.ean and state.san ~= state.ean then
6197         local d = node.new(DIR)
6198         d.dir = '+TLT'
6199         _, new = node.insert_before(head, state.san, d)

```

```

6200     if state.san == state.sim then state.sim = new end
6201     local d = node.new(DIR)
6202     d.dir = '-TLT'
6203     _, new = node.insert_after(head, state.ean, d)
6204     if state.ean == state.eim then state.eim = new end
6205 end
6206 new_state.san, new_state.ean = nil, nil
6207 return head, new_state
6208 end
6209
6210 -- TODO - \hbox with an explicit dir can lead to wrong results
6211 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6212 -- was s made to improve the situation, but the problem is the 3-dir
6213 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6214 -- well.
6215
6216 function Babel.bidi(head, ispar, hdir)
6217     local d -- d is used mainly for computations in a loop
6218     local prev_d = ''
6219     local new_d = false
6220
6221     local nodes = {}
6222     local outer_first = nil
6223     local inmath = false
6224
6225     local glue_d = nil
6226     local glue_i = nil
6227
6228     local has_en = false
6229     local first_et = nil
6230
6231     local ATDIR = luatexbase.registernumber'bbl@attr@dir'
6232
6233     local save_outer
6234     local temp = node.get_attribute(head, ATDIR)
6235     if temp then
6236         temp = temp % 3
6237         save_outer = (temp == 0 and 'l') or
6238                     (temp == 1 and 'r') or
6239                     (temp == 2 and 'al')
6240     elseif ispar then -- Or error? Shouldn't happen
6241         save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6242     else -- Or error? Shouldn't happen
6243         save_outer = ('TRT' == hdir) and 'r' or 'l'
6244     end
6245     -- when the callback is called, we are just _after_ the box,
6246     -- and the textdir is that of the surrounding text
6247     -- if not ispar and hdir ~= tex.textdir then
6248     --     save_outer = ('TRT' == hdir) and 'r' or 'l'
6249     -- end
6250     local outer = save_outer
6251     local last = outer
6252     -- 'al' is only taken into account in the first, current loop
6253     if save_outer == 'al' then save_outer = 'r' end
6254
6255     local fontmap = Babel.fontmap
6256
6257     for item in node.traverse(head) do
6258

```

```

6259 -- In what follows, #node is the last (previous) node, because the
6260 -- current one is not added until we start processing the neutrals.
6261
6262 -- three cases: glyph, dir, otherwise
6263 if item.id == GLYPH
6264     or (item.id == 7 and item.subtype == 2) then
6265
6266     local d_font = nil
6267     local item_r
6268     if item.id == 7 and item.subtype == 2 then
6269         item_r = item.replace -- automatic discs have just 1 glyph
6270     else
6271         item_r = item
6272     end
6273     local chardata = characters[item_r.char]
6274     d = chardata and chardata.d or nil
6275     if not d or d == 'nsm' then
6276         for nn, et in ipairs(ranges) do
6277             if item_r.char < et[1] then
6278                 break
6279             elseif item_r.char <= et[2] then
6280                 if not d then d = et[3]
6281                 elseif d == 'nsm' then d_font = et[3]
6282                 end
6283                 break
6284             end
6285         end
6286     end
6287     d = d or 'l'
6288
6289     -- A short 'pause' in bidi for mapfont
6290     d_font = d_font or d
6291     d_font = (d_font == 'l' and 0) or
6292             (d_font == 'nsm' and 0) or
6293             (d_font == 'r' and 1) or
6294             (d_font == 'al' and 2) or
6295             (d_font == 'an' and 2) or nil
6296     if d_font and fontmap and fontmap[d_font][item_r.font] then
6297         item_r.font = fontmap[d_font][item_r.font]
6298     end
6299
6300     if new_d then
6301         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6302         if inmath then
6303             attr_d = 0
6304         else
6305             attr_d = node.get_attribute(item, ATDIR)
6306             attr_d = attr_d % 3
6307         end
6308         if attr_d == 1 then
6309             outer_first = 'r'
6310             last = 'r'
6311         elseif attr_d == 2 then
6312             outer_first = 'r'
6313             last = 'al'
6314         else
6315             outer_first = 'l'
6316             last = 'l'
6317         end

```

```

6318         outer = last
6319         has_en = false
6320         first_et = nil
6321         new_d = false
6322     end
6323
6324     if glue_d then
6325         if (d == 'l' and 'l' or 'r') ~= glue_d then
6326             table.insert(nodes, {glue_i, 'on', nil})
6327         end
6328         glue_d = nil
6329         glue_i = nil
6330     end
6331
6332     elseif item.id == DIR then
6333         d = nil
6334         new_d = true
6335
6336     elseif item.id == node.id'glue' and item.subtype == 13 then
6337         glue_d = d
6338         glue_i = item
6339         d = nil
6340
6341     elseif item.id == node.id'math' then
6342         inmath = (item.subtype == 0)
6343
6344     else
6345         d = nil
6346     end
6347
6348     -- AL <= EN/ET/ES      -- W2 + W3 + W6
6349     if last == 'al' and d == 'en' then
6350         d = 'an'          -- W3
6351     elseif last == 'al' and (d == 'et' or d == 'es') then
6352         d = 'on'          -- W6
6353     end
6354
6355     -- EN + CS/ES + EN      -- W4
6356     if d == 'en' and #nodes >= 2 then
6357         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6358             and nodes[#nodes-1][2] == 'en' then
6359             nodes[#nodes][2] = 'en'
6360         end
6361     end
6362
6363     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
6364     if d == 'an' and #nodes >= 2 then
6365         if (nodes[#nodes][2] == 'cs')
6366             and nodes[#nodes-1][2] == 'an' then
6367             nodes[#nodes][2] = 'an'
6368         end
6369     end
6370
6371     -- ET/EN                  -- W5 + W7->1 / W6->on
6372     if d == 'et' then
6373         first_et = first_et or (#nodes + 1)
6374     elseif d == 'en' then
6375         has_en = true
6376         first_et = first_et or (#nodes + 1)

```

```

6377 elseif first_et then          -- d may be nil here !
6378     if has_en then
6379         if last == 'l' then
6380             temp = 'l'      -- W7
6381         else
6382             temp = 'en'     -- W5
6383         end
6384     else
6385         temp = 'on'        -- W6
6386     end
6387     for e = first_et, #nodes do
6388         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6389     end
6390     first_et = nil
6391     has_en = false
6392 end
6393
6394 -- Force mathdir in math if ON (currently works as expected only
6395 -- with 'l')
6396 if inmath and d == 'on' then
6397     d = ('TRT' == tex.mathdir) and 'r' or 'l'
6398 end
6399
6400 if d then
6401     if d == 'al' then
6402         d = 'r'
6403         last = 'al'
6404     elseif d == 'l' or d == 'r' then
6405         last = d
6406     end
6407     prev_d = d
6408     table.insert(nodes, {item, d, outer_first})
6409 end
6410
6411 outer_first = nil
6412
6413 end
6414
6415 -- TODO -- repeated here in case EN/ET is the last node. Find a
6416 -- better way of doing things:
6417 if first_et then          -- dir may be nil here !
6418     if has_en then
6419         if last == 'l' then
6420             temp = 'l'      -- W7
6421         else
6422             temp = 'en'     -- W5
6423         end
6424     else
6425         temp = 'on'        -- W6
6426     end
6427     for e = first_et, #nodes do
6428         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6429     end
6430 end
6431
6432 -- dummy node, to close things
6433 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6434
6435 ----- NEUTRAL -----

```



```

6436
6437 outer = save_outer
6438 last = outer
6439
6440 local first_on = nil
6441
6442 for q = 1, #nodes do
6443     local item
6444
6445     local outer_first = nodes[q][3]
6446     outer = outer_first or outer
6447     last = outer_first or last
6448
6449     local d = nodes[q][2]
6450     if d == 'an' or d == 'en' then d = 'r' end
6451     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
6452
6453     if d == 'on' then
6454         first_on = first_on or q
6455     elseif first_on then
6456         if last == d then
6457             temp = d
6458         else
6459             temp = outer
6460         end
6461         for r = first_on, q - 1 do
6462             nodes[r][2] = temp
6463             item = nodes[r][1] -- MIRRORING
6464             if Babel.mirroring_enabled and item.id == GLYPH
6465                 and temp == 'r' and characters[item.char] then
6466                 local font_mode = font.fonts[item.font].properties.mode
6467                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
6468                     item.char = characters[item.char].m or item.char
6469                 end
6470             end
6471         end
6472         first_on = nil
6473     end
6474
6475     if d == 'r' or d == 'l' then last = d end
6476 end
6477
6478 ----- IMPLICIT, REORDER -----
6479
6480 outer = save_outer
6481 last = outer
6482
6483 local state = {}
6484 state.has_r = false
6485
6486 for q = 1, #nodes do
6487
6488     local item = nodes[q][1]
6489
6490     outer = nodes[q][3] or outer
6491
6492     local d = nodes[q][2]
6493
6494     if d == 'nsm' then d = last end -- W1

```

```

6495   if d == 'en' then d = 'an' end
6496   local isdir = (d == 'r' or d == 'l')
6497
6498   if outer == 'l' and d == 'an' then
6499       state.san = state.san or item
6500       state.ean = item
6501   elseif state.san then
6502       head, state = insert_numeric(head, state)
6503   end
6504
6505   if outer == 'l' then
6506       if d == 'an' or d == 'r' then      -- im -> implicit
6507           if d == 'r' then state.has_r = true end
6508           state.sim = state.sim or item
6509           state.eim = item
6510       elseif d == 'l' and state.sim and state.has_r then
6511           head, state = insert_implicit(head, state, outer)
6512       elseif d == 'l' then
6513           state.sim, state.eim, state.has_r = nil, nil, false
6514       end
6515   else
6516       if d == 'an' or d == 'l' then
6517           if nodes[q][3] then -- nil except after an explicit dir
6518               state.sim = item -- so we move sim 'inside' the group
6519           else
6520               state.sim = state.sim or item
6521           end
6522           state.eim = item
6523       elseif d == 'r' and state.sim then
6524           head, state = insert_implicit(head, state, outer)
6525       elseif d == 'r' then
6526           state.sim, state.eim = nil, nil
6527       end
6528   end
6529
6530   if isdir then
6531       last = d      -- Don't search back - best save now
6532   elseif d == 'on' and state.san then
6533       state.san = state.san or item
6534       state.ean = item
6535   end
6536
6537 end
6538
6539 return node.prev(head) or head
6540 end
6541 </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.

```
6542 ⟨*nil⟩
6543 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
6544 \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.

```
6545 \ifx\l@nil\undefined
6546   \newlanguage\l@nil
6547   \@namedef{bbl@hyphendata@the\l@nil}{\{}}% Remove warning
6548   \let\bbl@elt\relax
6549   \edef\bbl@languages{% Add it to the list of languages
6550     \bbl@languages\bbl@elt{nil}{the\l@nil}{\{}}
6551 \fi
```

This macro is used to store the values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```
6552 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

The next step consists of defining commands to switch to (and from) the ‘nil’ language.

```
\captionnil
\datenil
6553 \let\captionnil\@empty
6554 \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.

```
6555 \ldf@finish{nil}
6556 ⟨/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`.

```

6557 \catcode\lbrack=1 % left brace is begin-group character
6558 \catcode\rbrack=2 % right brace is end-group character
6559 \catcode\hash=6 % hash mark is macro parameter character

```

```
6561 \openin 0 hyphen.cfg
6562 \ifeof0
6563 \else
6564   \let\input
```

```

6565 \def\input #1 {%
6566   \let\input\@
6567   \a hyphen.cfg
6568   \let\@undefined
6569 }
6570 \fi
6571 </bplain | bplain>

```

```
6572 \bplain\ a plain.tex
6573 \blplain\ a lplain.tex
```

```
6574 \bplain\def\fmtname{babel-plain}
6575 \blplain\def\fmtname{babel-lplain}
```

## 16.2 Emulating some L<sup>A</sup>T<sub>E</sub>X features

```

6576 <(<Emulate LaTeX>) ≡
6577 % == Code for plain ==
6578 \def\@empty{}
6579 \def\loadlocalcfg#1{%
6580   \openin0#1.cfg
6581   \ifeof0
6582     \closein0
6583   \else
6584     \closein0
6585     {\immediate\write16{*****}%
6586      \immediate\write16{* Local config file #1.cfg used}%
6587      \immediate\write16{*}%
6588     }
6589     \input #1.cfg\relax
6590   \fi
6591   \@endoflfd}

```

## 16.3 General tools

A number of  $\text{\LaTeX}$  macro's that are needed later on.

```

6592 \long\def\@firstofone#1{#1}
6593 \long\def\@firstoftwo#1#2{#1}
6594 \long\def\@secondoftwo#1#2{#2}
6595 \def\@nnil{\@nil}
6596 \def\@gobbletwo#1#2{}
6597 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
6598 \def\@star@or@long#1{%
6599   \@ifstar
6600   {\let\l@ngrel@x\relax#1}%
6601   {\let\l@ngrel@x\long#1}}
6602 \let\l@ngrel@x\relax
6603 \def\@car#1#2\@nil{#1}
6604 \def\@cdr#1#2\@nil{#2}
6605 \let\@typeset@protect\relax
6606 \let\protected@edef\edef
6607 \long\def\@gobble#1{}
6608 \edef\@backslashchar{\expandafter\@gobble\string\}
6609 \def\strip@prefix#1>{}
6610 \def\g@addto@macro#1#2{%
6611   \toks@\expandafter{#1#2}%
6612   \xdef#1{\the\toks@}}
6613 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
6614 \def\@nameuse#1{\csname #1\endcsname}
6615 \def\@ifundefined#1{%
6616   \expandafter\ifx\csname#1\endcsname\relax
6617     \expandafter\@firstoftwo
6618   \else
6619     \expandafter\@secondoftwo
6620   \fi}
6621 \def\@expandtwoargs#1#2#3{%
6622   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
6623 \def\zap@space#1 #2{%
6624   #1%
6625   \ifx#2\@empty\else\expandafter\zap@space\fi
6626   #2}
6627 \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}`.

```

6628 \ifx\@preamblecmds\@undefined
6629   \def\@preamblecmds{}
6630 \fi
6631 \def\@onlypreamble#1{%
6632   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
6633     \@preamblecmds\do#1}}
6634 \@onlypreamble\onlypreamble

```

Mimick  $\text{\LaTeX}$ 's `\AtBeginDocument`; for this to work the user needs to add `\begindocument` to his file.

```

6635 \def\begindocument{%
6636   \@begindocumenthook
6637   \global\let\@begindocumenthook\@undefined
6638   \def\do##1{\global\let##1\@undefined}%
6639   \@preamblecmds
6640   \global\let\do\noexpand}
6641 \ifx\@begindocumenthook\@undefined
6642   \def\@begindocumenthook{}

```

```

6643 \fi
6644 \@onlypreamble\@begindocumenthook
6645 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

We also have to mimick LATEX's \AtEndOfPackage. Our replacement macro is much simpler; it stores
its argument in \@endofldf.

6646 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
6647 \@onlypreamble\AtEndOfPackage
6648 \def\@endofldf{}
6649 \@onlypreamble\@endofldf
6650 \let\bbl@afterlang\@empty
6651 \chardef\bbl@opt@hyphenmap\z@

```

L<sup>A</sup>T<sub>E</sub>X needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default. There is a trick to hide some conditional commands from the outer \ifx. The same trick is applied below.

```

6652 \catcode`\&=\z@
6653 \ifx&\if@files\@undefined
6654 \expandafter\let\csname if@files\expandafter\endcsname
6655 \csname iffalse\endcsname
6656 \fi
6657 \catcode`\&=4

```

Mimick L<sup>A</sup>T<sub>E</sub>X's commands to define control sequences.

```

6658 \def\newcommand{\@star@or@long\new@command}
6659 \def\new@command#1{%
6660 \@testopt{\@newcommand#1}0}
6661 \def\@newcommand#1[#2]{%
6662 \@ifnextchar [{\@xargdef#1[#2]}%
6663 {\@argdef#1[#2]}}
6664 \long\def\@argdef#1[#2]#3{%
6665 \@yargdef#1\@ne{#2}{#3}}
6666 \long\def\@xargdef#1[#2][#3]#4{%
6667 \expandafter\def\expandafter#1\expandafter{%
6668 \expandafter\@protected@testopt\expandafter #1%
6669 \csname\string#1\expandafter\endcsname{#3}}%
6670 \expandafter\@yargdef \csname\string#1\endcsname
6671 \tw@{#2}{#4}}
6672 \long\def\@yargdef#1#2#3{%
6673 \@tempcnta#3\relax
6674 \advance \@tempcnta \@ne
6675 \let\@hash\relax
6676 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
6677 \@tempcntb #2%
6678 \@whilenum\@tempcntb <\@tempcnta
6679 \do{%
6680 \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
6681 \advance\@tempcntb \@ne}%
6682 \let\@hash@###
6683 \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
6684 \def\providecommand{\@star@or@long\provide@command}
6685 \def\provide@command#1{%
6686 \begingroup
6687 \escapechar\m@ne\xdef\@gtempa{\string#1}%
6688 \endgroup
6689 \expandafter\ifundefined\@gtempa
6690 {\def\reserved@a{\new@command#1}}%
6691 {\let\reserved@a\relax
6692 \def\reserved@a{\new@command\reserved@a}}%
6693 \reserved@a}%

```

```

6694 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
6695 \def\declare@robustcommand#1{%
6696   \edef\reserved@a{\string#1}%
6697   \def\reserved@b{#1}%
6698   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
6699   \edef#1{%
6700     \ifx\reserved@a\reserved@b
6701       \noexpand\x@protect
6702       \noexpand#1%
6703     \fi
6704     \noexpand\protect
6705     \expandafter\noexpand\csname
6706       \expandafter\@gobble\string#1 \endcsname
6707   }%
6708   \expandafter\new@command\csname
6709     \expandafter\@gobble\string#1 \endcsname
6710 }
6711 \def\x@protect#1{%
6712   \ifx\protect\@typeset@protect\else
6713     \@x@protect#1%
6714   \fi
6715 }
6716 \catcode`\&=\z@ % Trick to hide conditionals
6717 \def\@x@protect#1&fi##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`.

```

6718 \def\bbl@tempa{\csname newif\endcsname&ifin@}
6719 \catcode`\&=4
6720 \ifx\in@\@undefined
6721   \def\in@#1#2{%
6722     \def\in@@##1#1##2##3\in@@{%
6723       \ifx\in@@##2\in@false\else\in@true\fi}%
6724     \in@@#2#1\in@\in@@}
6725 \else
6726   \let\bbl@tempa\@empty
6727 \fi
6728 \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).

```

6729 \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.

```

6730 \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.

```

6731 \ifx\@tempcnta\@undefined
6732   \csname newcount\endcsname\@tempcnta\relax
6733 \fi
6734 \ifx\@tempcntb\@undefined
6735   \csname newcount\endcsname\@tempcntb\relax
6736 \fi

```

To prevent wasting two counters in L<sup>A</sup>T<sub>E</sub>X 2.09 (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```

6737 \ifx\bye\undefined
6738   \advance\count10 by -2\relax
6739 \fi
6740 \ifx\@ifnextchar\undefined
6741   \def\@ifnextchar#1#2#3{%
6742     \let\reserved@d=#1%
6743     \def\reserved@a{#2}\def\reserved@b{#3}%
6744     \futurelet\@let@token\@ifnch}
6745   \def\@ifnch{%
6746     \ifx\@let@token\@sptoken
6747       \let\reserved@c\@xifnch
6748     \else
6749       \ifx\@let@token\reserved@d
6750         \let\reserved@c\reserved@a
6751       \else
6752         \let\reserved@c\reserved@b
6753     \fi
6754   \fi
6755   \reserved@c}
6756   \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
6757   \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
6758 \fi
6759 \def\@testopt#1#2{%
6760   \@ifnextchar[#{1}{#1[#2]}}
6761 \def\@protected@testopt#1{%
6762   \ifx\protect\@typeset@protect
6763     \expandafter\@testopt
6764   \else
6765     \@x@protect#1%
6766   \fi}
6767 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
6768   #2\relax}\fi}
6769 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
6770   \else\expandafter\@gobble\fi{#1}}

```

## 16.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain T<sub>E</sub>X environment.

```

6771 \def\DeclareTextCommand{%
6772   \@dec@text@cmd\providecommand
6773 }
6774 \def\ProvideTextCommand{%
6775   \@dec@text@cmd\providecommand
6776 }
6777 \def\DeclareTextSymbol#1#2#3{%
6778   \@dec@text@cmd\chardef#1{#2}#3\relax
6779 }
6780 \def\@dec@text@cmd#1#2#3{%
6781   \expandafter\def\expandafter#2%
6782     \expandafter{%
6783       \csname#3-cmd\expandafter\endcsname
6784       \expandafter#2%
6785       \csname#3\string#2\endcsname
6786     }%
6787 %   \let\@ifdefinable\@rc@ifdefinable
6788   \expandafter#1\csname#3\string#2\endcsname

```



```

6789 }
6790 \def\@current@cmd#1{%
6791   \ifx\protect\@typeset@protect\else
6792     \noexpand#1\expandafter\@gobble
6793   \fi
6794 }
6795 \def\@changed@cmd#1#2{%
6796   \ifx\protect\@typeset@protect
6797     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
6798       \expandafter\ifx\csname ?\string#1\endcsname\relax
6799         \expandafter\def\csname ?\string#1\endcsname{%
6800           \@changed@x@err{#1}%
6801         }%
6802       \fi
6803     \global\expandafter\let
6804       \csname\cf@encoding\string#1\expandafter\endcsname
6805       \csname ?\string#1\endcsname
6806     \fi
6807     \csname\cf@encoding\string#1%
6808       \expandafter\endcsname
6809   \else
6810     \noexpand#1%
6811   \fi
6812 }
6813 \def\@changed@x@err#1{%
6814   \errhelp{Your command will be ignored, type <return> to proceed}%
6815   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}%
6816 \def\DeclareTextCommandDefault#1{%
6817   \DeclareTextCommand#1?%
6818 }
6819 \def\ProvideTextCommandDefault#1{%
6820   \ProvideTextCommand#1?%
6821 }
6822 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
6823 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
6824 \def\DeclareTextAccent#1#2#3{%
6825   \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
6826 }
6827 \def\DeclareTextCompositeCommand#1#2#3#4{%
6828   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
6829   \edef\reserved@b{\string##1}%
6830   \edef\reserved@c{%
6831     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
6832   \ifx\reserved@b\reserved@c
6833     \expandafter\expandafter\expandafter\ifx
6834       \expandafter\@car\reserved@a\relax\relax\@nil
6835     \@text@composite
6836   \else
6837     \edef\reserved@b##1{%
6838       \def\expandafter\noexpand
6839         \csname#2\string#1\endcsname###1{%
6840           \noexpand\@text@composite
6841             \expandafter\noexpand\csname#2\string#1\endcsname
6842             ###1\noexpand\empty\noexpand\@text@composite
6843             {##1}%
6844         }%
6845       }%
6846     \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
6847   \fi

```

```

6848 \expandafter\def\csname\expandafter\string\csname
6849 #2\endcsname\string#1-\string#3\endcsname{#4}
6850 \else
6851 \errhelp{Your command will be ignored, type <return> to proceed}%
6852 \errmessage{\string\DeclareTextCompositeCommand\space used on
6853 inappropriate command \protect#1}
6854 \fi
6855 }
6856 \def\@text@composite#1#2#3\@text@composite{%
6857 \expandafter\@text@composite@x
6858 \csname\string#1-\string#2\endcsname
6859 }
6860 \def\@text@composite@x#1#2{%
6861 \ifx#1\relax
6862 #2%
6863 \else
6864 #1%
6865 \fi
6866 }
6867 %
6868 \def\@strip@args#1:#2-#3\@strip@args{#2}
6869 \def\DeclareTextComposite#1#2#3#4{%
6870 \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
6871 \bgroup
6872 \lccode\@=#4%
6873 \lowercase{%
6874 \egroup
6875 \reserved@a @%
6876 }%
6877 }
6878 %
6879 \def\UseTextSymbol#1#2{#2}
6880 \def\UseTextAccent#1#2#3{}
6881 \def\@use@text@encoding#1{}
6882 \def\DeclareTextSymbolDefault#1#2{%
6883 \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
6884 }
6885 \def\DeclareTextAccentDefault#1#2{%
6886 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
6887 }
6888 \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.

```

6889 \DeclareTextAccent{"}{OT1}{127}
6890 \DeclareTextAccent{'}{OT1}{19}
6891 \DeclareTextAccent{^}{OT1}{94}
6892 \DeclareTextAccent`}{OT1}{18}
6893 \DeclareTextAccent~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for `PLAIN TEX`.

```

6894 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
6895 \DeclareTextSymbol{\textquotedblright}{OT1}{93}
6896 \DeclareTextSymbol{\textquoteleft}{OT1}{96}
6897 \DeclareTextSymbol{\textquoteright}{OT1}{97}
6898 \DeclareTextSymbol{\i}{OT1}{16}
6899 \DeclareTextSymbol{\ss}{OT1}{25}

```

For a couple of languages we need the  $\text{\LaTeX}$ -control sequence `\scriptsize` to be available. Because plain  $\text{T}_{\text{E}}\text{X}$  doesn't have such a sophisticated font mechanism as  $\text{\LaTeX}$  has, we just `\let` it to `\sevenrm`.

```

6900 \ifx\scriptsize\@undefined
6901   \let\scriptsize\sevenrm
6902 \fi
6903 % End of code for plain
6904 <</Emulate LaTeX>>

A proxy file:
6905 <plain>
6906 \input babel.def
6907 </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  $\text{\LaTeX}$  styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The  $\text{\TeX}$ book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport,  *$\text{\LaTeX}$ , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in:  $\text{\TeX}$ hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Hubert Partl, *German  $\text{\TeX}$* , *TUGboat* 9 (1988) #1, p. 70–72.
- [10] Joachim Schrod, *International  $\text{\LaTeX}$  is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [11] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using  $\text{\LaTeX}$* , Springer, 2002, p. 301–373.
- [12] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).